1-2020

Root Failure Analysis in Meshed Tree Networks

Shashank Rudroju
sr1632@rit.edu

ROOT FAILURE ANALYSIS IN MESHED TREE NETWORKS

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Leonid Reznik, Supervisor

Dr. Nirmala Shenoy, Reader

Dr. H.B. Acharya, Observer

ROOT FAILURE ANALYSIS IN MESHED TREE NETWORKS

by

Shashank Rudroju

THESIS

Presented to the Faculty of the Department of Computer Science

Golisano College of Computer and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

Rochester Institute of Technology

January 2020

Dedicated to all the researchers around the world.

Acknowledgments

This work would not have been possible without the support and wisdom of many people in my life. Most importantly, I wish to thank Dr. Nirmala Shenoy who guided me throughout my Master's journey and gave me a great opportunity to do research under her. Professor taught me Computer Networking and many important life lessons. To my committee members Dr. Leon Reznik and Dr. H.B. Acharya. To my family, friends and my academic advisor Cindy who encouraged and motivated me in finishing my thesis. Special mention to my friend Saarika for being there when I needed the most ...

Abstract

ROOT FAILURE ANALYSIS IN MESHED TREE NETWORKS

Shashank Rudroju, M.S.
Rochester Institute of Technology, 2020

Supervisor: Dr. Leonid Reznik

Mesh topologies play a vital role in switched networks. Broadcast storms due to the loops in Mesh Networks are a major concern in switched networks. Logical spanning trees are constructed using algorithms like spanning tree algorithm to avoid loops and hence address the broadcast storm problem. However, in the event of a topology change or a link failure in the network, it takes time to converge and construct new spanning tree to forward frames. Link State routing[18][26] and other protocols like Rapid Spanning Tree protocol[2][19] were introduced to address the problems of high convergence times in the basic spanning tree protocol(STP)[1][4] in the event of network component failures. A much efficient and advanced approach was offered with Mesh Tree Protocol based on the Mesh Tree Algorithm. Mesh Tree Protocol constructs multiple tree branches from a single root and quickly falls back to an alternate path or switch in case of link or switch failures. This cuts down the convergence delays considerably.

The Mesh Tree Protocol based on the Mesh Tree Algorithm is currently under development as an IEEE standard[28][27]. Other major changes in the MTP compared to the already existing protocols is that the root is manually assigned instead of using the root election procedure. This will cut down the delays during instantiation of the protocol but also has risk concerning the action of the protocol if the manually assigned root fails. To address this concern, an enhancement to the Mesh Tree protocol is being researched in this thesis. The idea is to implement a Multiple Meshed Tree algorithm where meshed trees will be constructed from multiple roots. This thesis introduces root redundancy in the Mesh Tree Protocol and will be assessed for performance improvements on root failures in comparison with Rapid Spanning Tree Protocol (RSTP) which re-elects a root switch on the current root switch failure.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Convergence latency in switched networks is primarily attributed to the switching operations that happen at layer 2 or the data link layer in the network on topology changes. Network devices and links are prone to failures and this cannot be avoided, hence it is important to ensure quick recovery to alleviate service disruptions on failures. To address this serious problem, in networks it is a common practice to add redundant physical links. This practice leads to loops that will cause broadcast storms and eventually lead to network crash. In order to avoid such network loops, logical spanning trees are constructed, and various loop avoidance protocols have evolved over time. Some of the loop avoidance protocols are Spanning Tree Protocol (STP)[1], Rapid Spanning Tree Protocol (RSTP[2]. STP exhibited unacceptable convergence times due to the delay in constructing a new spanning tree any time there is a topology change. This lead to the development of Rapid STP (RSTP). RSTP still did not address the needs of high-speed networks such as data centers and service provider networks, which forced the research community to adopt the very complex link state routing at layer 2. During the convergence process, there is no guarantee of frame delivery, hence it is very critical to achieve quick convergence. In high-speed gigabit networks, network performance is adversely impacted if there is a loss of communication for milliseconds or microseconds. The failure detection time

and protocol recovery time both contribute to the overall convergence of a protocol. Link layer failure detection can be fast, however, after failure detection, the protocol takes time to recover and stabilize its frame forwarding. This delay in recovering or the protocol failure recovery time is a direct measure of the protocol's performance. Hence a novel approach to achieving fast failure recovery is important.

In RSTP, a root switch carries a more traffic operational overhead than other switches. However, root election is based on switch Bridge ID, comprising of Bridge priority (which can be administratively set) and MAC address of the switch. If default bridge priority is used, then the metric for root election is the MAC address, and the switch with the lowest MAC address is elected root. However, this switch may not be the switch with the best processing capacity in the network. This introduces a potential bottleneck as this switch handles all broadcast, multicast and unicast traffic between branches stemming directly from the root. So, often a switch with the highest capacity is administratively biased by setting its priority to win the root election. It would seem redundant to bias root election and still go through the process of root election. The process of root election in STP and RSTP protocols and how root failure effects the performance of these protocols is discussed in detail in further sections.

Further research in the area to cut down the performance overhead of root election paved the path for a novel Meshed Tree Algorithm (MTA)[21] [23] that supports multiple tree branches from a single root to cut down convergence delays on link or switch failures. Unlike Spanning Tree Protocol where a single spanning tree is constructed, the Mesh Tree Algorithm allows for multiple trees to be constructed

from a single root and thus non-root switches and network segments reside in multiple tree branches. In the event of failure of one branch in a tree, another branch is in readiness to take over. Network-wide dissemination of change information for path re-calculation is not required as a result. This protocol has been in development for a while and made considerable progress in the recent years. Detailed implementation and the working of the Mesh Tree Protocol is published in various papers.

Unlike the root election process in loop-avoidance protocols such as RSTP, in MTP the root node of the meshed trees is designated to one of the meshed tree switches, which bypasses a root election altogether. This design decision was made due to the fact that administrators in an enterprise network usually bias the root election by setting a priority field so that a switch with better processing capacity is elected as the root. Even when the priority of a bridge ID (BID) is manually set low enough that a particular bridge will win the root election, the election process still has to occur so that the other switches in the topology recognize the root bridge's superiority, which adds additional latency to the initial convergence as well as the additional overhead in control frames sent between nodes so that they have enough information to make a proper decision. In MTP, once the desired root switch has been designated that role (currently a command line argument when starting the protocol implementation), the other switches that wish to participate in MTP simply wait for switch to start the process of building the meshed tree.

Even though Mesh Tree Protocol provides faster convergence times and simpler complexity, it still has the risk of the root failure. Multi-Meshed Trees (MMT), developed as a part of this thesis, are an extension of Meshed Trees where multiple

roots support their own meshed trees. By introducing root redundancy in the Mesh Tree Protocol, we avoid the root election process in the event of root failures. Root redundancy can cut down the cost of root election in the event of root failure and the protocol can fall back to a secondary root and rely on the meshed tree from the secondary root for frame forwarding. Consider the case where two switches were assigned to be roots in a network. One is the primary root, the other is the secondary root and each is assigned a unique tree ID, say 1 and 2. Switches hearing the IDs advertised from the two roots, will join the meshed trees from both the roots. Accordingly, they will store the tree related information in two tables. The broadcast, tree will be maintained for meshed trees originating from each of the roots. In case of primary root failure, the secondary root will take over and the tree corresponding to the secondary root will be used for frame forwarding.

1.1 Goals and Objectives

The main aim of the thesis is to develop an enhancement for the Mesh Tree Protocol by adopting MMTP to introduce root redundancy and compare the performance of the protocol with RSTP, that is currently in use in customer VLANs, in the case of root failures. We show the significantly improved performance in each of the root failures for different network topologies.

Global Environment for Network Innovation (GENI)[17] is the testbed that is used for testing the MTP (Meshed Tree Protocol) based on the Multi Mesh Tree Algorithm(MMTA). GENI provides us with a Lab as a Service (LaaS) kind of environment where we can create hosts with virtual interfaces and construct multiple

topologies with any choice of base operating system for these hosts. For the study of this thesis, the author used GENI and various GENI sites in the US for creating topologies and setting up protocols on them.

To implement MTP based on MMTA, the author designated the primary and secondary roots initially when the MTP is started in the switches i.e. the two roots are chosen manually by the author, one as the primary and the other as secondary. This gives us the advantage of deciding the roots based on the capacity of the switches. This thesis was to provide proof of concept for Multi Mesh Tree Algorithm based Mesh Tree Protocol, hence the implementation is limited to a primary root and a secondary root. The protocol is tested on topologies in the scenario where the primary root fails, and the protocol goes through a fail-over to the secondary root and the corresponding mesh tree from the secondary root is used.

The author set up similar topologies in GENI for MMTA based Mesh Tree Protocol and RSTP, failed the primary root and assessed the convergence times.

To avoid setting up the protocol and collecting the metrics manually on all the nodes for various topologies, automation scripts are used. These automation scripts will connect to all the nodes in a topology and set up the protocol and will also aid the process of collecting performance metrics. The automation scripts that were originally developed to set up MTP are modified to work with the new enhanced Multi Mesh Tree Protocol. These automation scripts are written in Python and Bash. The original MTP implementation code that is written in C is taken and root redundancy is introduced as a part of the code.

Chapter 2

Background

Currently, Service Providers (SP), Backbone Providers(BP) and data centers use high-performance switched networks that provide loop avoidance and Virtual LANs. Numerous loop avoidance algorithms were introduced in the past that block the ports of the switches logically. Spanning Tree Protocol[1, 22, 4] and Rapid Spanning Tree Protocol are two such loop avoidance protocols that create a logical tree for frame forwarding.

2.1 Importance of root in loop avoidance protocols

The root plays the most vital role in loop avoidance protocols. The root bridge acts as the central and reference point for calculating a loop free tree from a given meshed network topology. The first step in the process of initialization for protocols STP and RSTP is root election. The following sections will explain the importance of root and the consequences of a root failure in both the protocols STP and RSTP.

2.2 Root failure convergence in Spanning Tree Protocol

Spanning tree protocol initialization works in the following three steps.

1. Selecting the root bridge

2. Identifying the root ports on non-root bridges

3. Identifying the designated ports and blocked ports in all switches

Since the convergence time in the case of a switch going down in the network for the traditional STP is already explained in many papers[22] as 50 seconds, this thesis focuses only on the scenario where the root bridge fails. In order to understand the consequences of what happens after a root bridge fails, there is a need to understand the process in which the root is elected first.

2.2.1 Election of the root bridge in STP

Every switch in a network has a unique identity called Bridge ID. It is an 8-byte field comprising two parts according to the original 801.2D standard[1]. The initial segment of the Bridge ID is a 2 byte configurable field called as the Bridge Priority field and the subsequent part is the unique MAC address of the switch. The combination of these two fields ensures a unique value for the Bridge ID.

The decision on who will become the Root Bridge is taken after an exchange of several STP messages between the switches, followed by an election. These STP messages are called Bridge Protocol Data Units or simply BPDUs. Each BPDU consists of various fields and Table 2.1 characterizes each field. These fields are important to know to understand the Root bridge election process.

The criteria for the root election is simple. Among a set of switches that are connected in a network, the switch with the minimal Bridge ID will be elected as

Field	Description
Root Bridge ID	Bridge ID of the switch assumed to be the root switch by this BPDU
Sender's Bridge ID	Bridge ID of the switch that is sending this BPDU
Cost to the Root Bridge	The cost calculated from this switch to the current root
Timer values	Forward delay, Max Age, Hello timers

Table 2.1: BPDU Message Fields

the Root Bridge. As the Bridge ID comprises both the Bridge Priority field and the MAC address, the first comparison is among the Bridge Priorities of the switches. The lowest Bridge priority switch becomes the Root Bridge. On the off chance that there is a tie between two switches having the same priority value, the MAC addresses are compared, and the switch with the lowest MAC address is chosen as the Root Bridge which is shown in Figure 2.1.

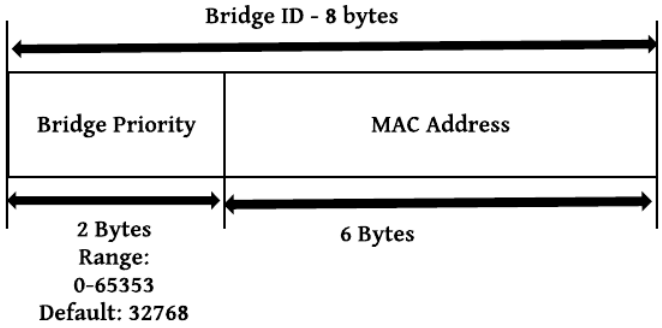

Figure 2.1: Bridge Priority

The election process for STP Root bridge begins with each switch creating

their own configuration BPDU. Initially, all the switches assign the root bridge ID field in the BPDU as its own bridge ID. The cost field in the configuration BPDU is initially set as 0, considering the fact that there is no cost to reach itself. These configuration BPDUs are exchanged among all switches. Each switch will continue to be a root bridge as long as they don't receive any BPDU that has a bridge priority lower than itself. Once a switch receives a BPDU that has a lower priority than itself, it updates the Root Bridge field in the BPDU with that value. All the switches will update their root bridge ID in the BPDUs that they receive and eventually agree on a common switch that will be the root bridge.

MaxAge	10 x Hello (20 Seconds)
Forward Delay	Listening 15 Seconds Learning 15 Seconds Forwarding

Table 2.2: STP timers

Following are some disadvantages with the traditional STP approach:

1. Every switch is assigned the same priority(32768) by design. Unless a better(lower) priority is manually configured to a switch, this approach will automatically elect the Root Bridge for you. This might result in the election of an edge switch that is very small with weak uplinks. Since Root Bridge in a spanning tree participates in the entire traffic flow in the network and is the logical center, a weak Root bridge selection will make the network less secure and less stable.

2. In the event of a connectivity failure in which a random switch is off the network

or a new switch is added to the network, STP has to re-calculate the path to the root for each switch. This healing process itself takes considerable time but if the root bridge is down, it constitutes a major topology change. A new root has to be re-elected and during the whole time the entire network will freeze and packets cannot be forwarded.

3. All the proprietary enhancements for STP[5][3] are developed to improve the converge under some conditions where there is a topology change in the network. In case of root failures, Spanning tree protocol has to go through the entire process of root election and building the Spanning tree by assigning the root ports, designated ports and the blocked ports

4. Spanning tree Portfast is a proprietary enhancement to Spanning tree protocol to help speed up network convergence on only access ports. PortFast causes a port to enter the spanning tree forwarding state immediately bypassing the listening and learning states. PortFast should only be used when connecting a single end station to a switch port. It should not be enabled on a port that is connected to another networking device such as a switch, otherwise it would cause loops in the network.

Mesh Tree Protocol avoids the whole process of root election, hence in this thesis, the protocol leverages this advantage of the Mesh Tree Protocol by selecting two roots and in case of a root failure, the protocol will failover to the secondary tree. A detailed explanation of the Multi Mesh Tree Protocol is discussed in the further sections.

The original 802.1D standard Spanning tree protocol was designed at a time when the networks considered the convergence time of around a minute acceptable or satisfactory. But with the advent of high speed applications and the need for faster convergence, enhancements[10] to the original STP were made. Enhancements like UplinkFast, BackboneFast and PortFast were designed to speed up the convergence times but they were only proprietary and implementing them would need extra care in making these configuration changes. The risk of misconfiguration was still there as each enhancement had its own conditions. The evolution of the 802.1D STP lead to the development of the Rapid Spanning Tree Protocol as an IEEE 802.1w standard. Most of the enhancements that are developed for the original Spanning Tree Protocol are bundled in the Rapid Spanning Tree Protocol. The following section explains the working of the RSTP in general and its convergence mechanism in case of root failures.

2.3 Root failure convergence in Rapid Spanning Tree Protocol(RSTP)

Even though there were improvements to the original 802.1D Spanning tree protocol with features like portFast, uplinkFast and the backboneFast, there was still a lot of room for improvement because the demand on the network to converge faster was growing. The primary goal for the development of RSTP is faster convergence but this is not achieved by just adjusting the timer values. There are a lot of changes and enhancements that are introduced in RSTP[19] for topology change convergence but retains the concept of the root bridge and the root election procedure. In case of

a root failure, similar to what was seen in the STP, the network has to still undergo the process of electing the new root and all the switches in the network have to learn about the new root. This procedure is time consuming and also requires a lot of BPDU message exchanges. In addition to the regular root election time, the new enhancements in the RSTP also pose a couple of new threats in some special conditions usually referred to as the count to infinity problem and the race condition problem[13]. Before discussing these problems, there is a need to understand the basic functioning of the RSTP and its synchronization process. In the next 2 sections, the similarities and differences between STP and RSTP are discussed.

During the development of RSTP, some of the concepts that are used in STP are retained even in RSTP.

- The concept of root bridge is still valid in RSTP.
- There is no change in the root election procedure for RSTP in comparison to the STP.
- Hello BPDUs are still used for communication between the switches

The major changes or enhancements that are introduced in RSTP in comparison to the STP are as follows:

1. **Port Roles:**

   In the Figure 2.2, both alternate port and backup port are in the blocked state but they function differently. Alternate port is similar to blocked port in the

Spanning Tree Protocol (IEEE 802.1D)	Rapid Spanning Tree Protocol (IEEE 802.1W)
Root Port	Root Port
Designated Port	Designated Port
Blocked Port	Alternate Port
	Backup Port

Figure 2.2: Comparing 802.1d and 802.1w Port Roles

standard STP protocol . It is the second best root port for a switch to the root bridge. In case the link to the current root port fails, the alternate port will take over. Backup port is used when there is a redundant port to the designated port. If there is already a designated port forwarding to a segment, then the other port is determined as the backup port. In simpler terms, an alternate port is the second best root port and a backup port is the second best designated port connecting to another switch that is not a root bridge. Edge port is another new port that is introduced in RSTP but it does not really participate in the RSTP process as they do not create any switching loops. An edge port is directly connected to end device or a client workstation. Addition or removal of the edge ports will not constitute a topology change for RSTP.

2. **Port States:**

   As shown in the Figure 2.3, Discarding state is when the frames are dropped and no addresses are learned. Some of the scenarios where a port enters discarding

STP Port State	Equivalent RSTP Port State
Disabled	Discarding
Blocking	Discarding
Listening	Discarding
Learning	Learning
Forwarding	Forwarding

Figure 2.3: Comparing 802.1d and 802.1w Port States

state are link down, blocking or during the synchronization process. Learning state is when the frames are dropped, but addresses are learned. Forwarding state is when the frames start forwarding through that port.

3. **RSTP Link Types:**

   RSTP introduces a new concept of Link types. Links in RSTP are categorised into two types. If a link is a full duplex or has a direct connection to another switch it is called a point-to-point link. If the link is half duplex, generally a link connected to a hub, it is called a shared link. Point-to-point links play an important role in the RSTP synchronization process that is going to be covered in the next section.

4. **Timers:**

   To achieve faster convergence in case of topology changes, RSTP made some changes to the way timers were used in the standard STP. In order for STP to

respond to a topology change, it waits for Max Age time which is 20 seconds. RSTP detects a topology change in 3 missed hello messages which is usually 6 seconds or less. The state transition from a blocked state to a forwarding state takes about 2 forward delay times in STP which is 30 seconds. For RSTP, there is no more forward delay timer to transition a port to forwarding. RSTP does not depend on the timers anymore for transition, in fact it used a new approach to handle convergence in the network. This new approach is through a process called as synchronization. The process is explained in the next section.

5. **RSTP Synchronization:**

   In a standard STP, when a switch in the network detects a topology change, it is first propagated to the root bridge via Topology Change Notification(TCN) BPDU messages. The root bridge then sends out the TCN BPDU with to every other switch in the network.

   But in the RSTP, any detected topology change notification is sent out to all the switches as a one step process. The initiator of the topology change floods this information across the network. The additional overhead of notifying the root bridge is avoided in RSTP.

In the standard STP, BPDU messages originate at the root bridge and are relayed across the network where as in RSTP each switch originates its own BPDU message and propagates them among their neighbors. The BPDU message was also modified in RSTP and is used in a more effective way. The flags field in the BPDU message is modified as shown in the Figure 2.4.

IEEE 802.1D

Bit	Function
0	Topology Change
1	Unused
2	Unused
3	Unused
4	Unused
5	Unused
6	Unused
7	Topology Change Acknowledgement

IEEE 802.1W

Bit	Function
0	Topology Change
1	Proposal
2-3 00 01 10 11	Port Role: Unknown Alternative Root Port Designated Port
4	Learning
5	Forwarding
6	Agreement
7	Topology Change Acknowledgement

Figure 2.4: Comparing 802.1d and 802.1w bits in the flags field of the BPDU

In the standard STP, only 2 of the 8 bits in the flags field of the BPDU was utilized. RSTP uses all the 8 bits of the flags field.

Let us look at a small topology with 3 switches that are connected using point-to-point links to understand RSTP synchronization process in Figure 2.5.

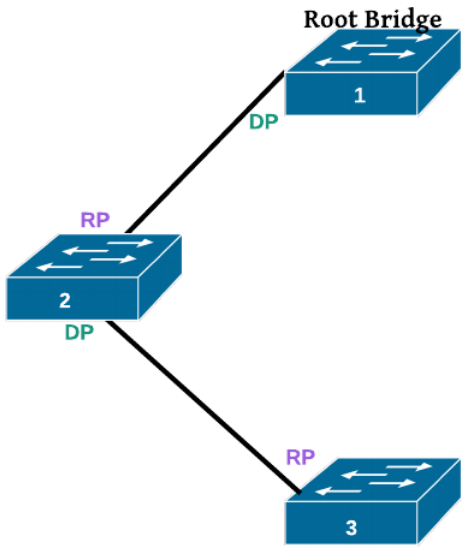

Figure 2.5: STEP-1: RSTP Synchronization Process

Switch-1 is the root bridge and root ports and designated ports are correctly assigned. Now suppose that a new switch, switch 4 is added to the network topology in between switch-1 and switch-3 as follows in Figure 2.6. This would cause a topology change and let us look at how RSTP reacts to this topology using the proposal/agreement mechanism in Figure 2.7.

First, the two ports at the end of the link connecting switch-1 and switch-4 are put into discarding state and start their negotiation. Switch-1 sends a BPDU

message with the its information and also including a proposal message in it. The proposal bit is set and the BPDU consists of other

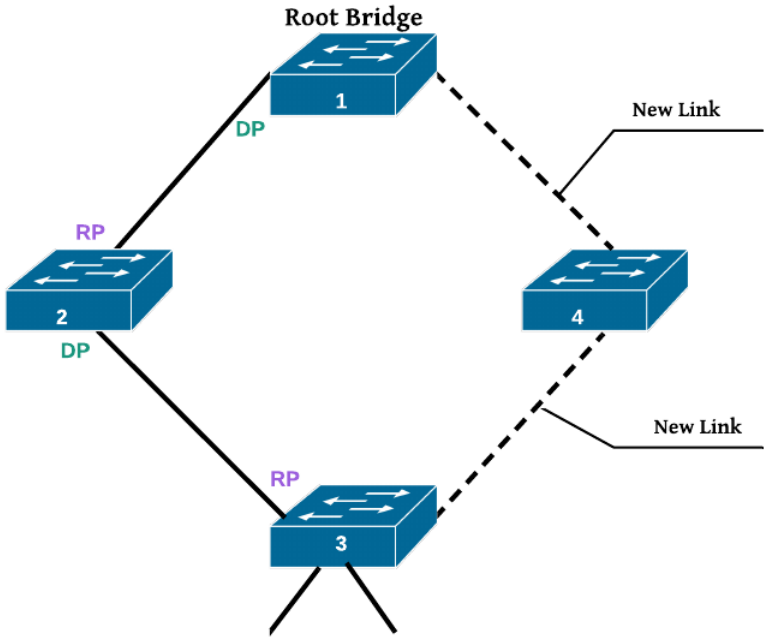

Figure 2.6: STEP-2: RSTP Synchronization Process

information and it translates to a message like "Hello, I am the root bridge and I propose that my port is a designated port and your port should be a root port". When the port on the switch-4 side that is currently in discarding state receives such a proposal, it checks if the BPDU received is superior. If the received BPDU is superior, like in our case, before it sends an agreement, switch-4 must block all the non-edge ports except the port on which it received the BPDU to avoid forming loops. After the agreement is sent, the receiving port is marked as the root port and

the sender's port is marked as designated port and both the ports are transitioned to forwarding state. If received BPDU is not superior, it rejects the BPDU and sets the senders port as alternate port. This process is called Synchronization.

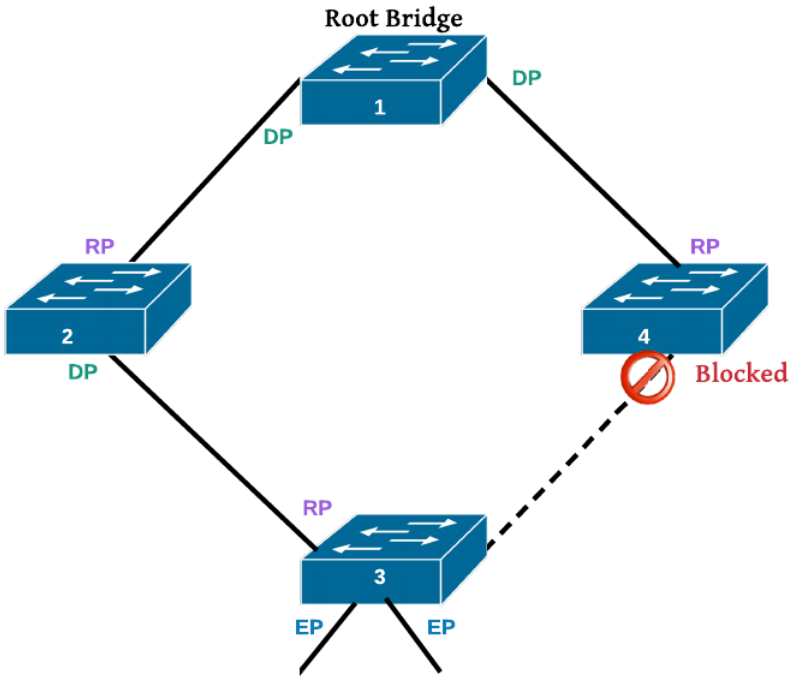

Figure 2.7: STEP-3: RSTP Synchronization Process

After the synchronization is completed for the link between switch-1 and switch-4, switch-4 will now sends out proposal messages on all the previously blocked ports connected to other switches. In our case, the ports on either sides of the link between switch switch-4 and switch-3 are moved to discarding state first as shown in Figure 2.8.

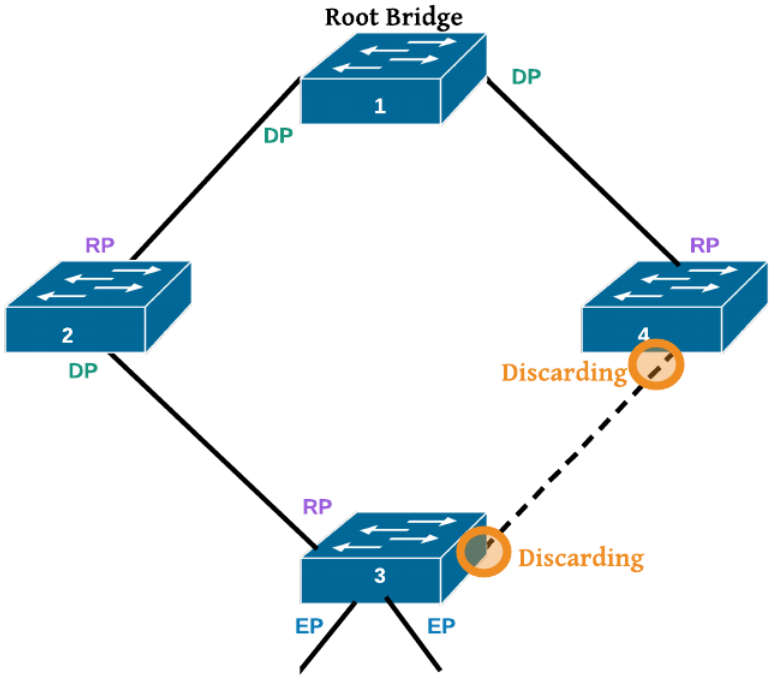

Figure 2.8: STEP-4: RSTP Synchronization Process

Then switch-4 sends its proposal to switch-3. When switch-3 receives the proposal and it verifies that switch-4's proposal is superior and acceptable it agrees that switch-3's best path to the root bridge is through switch-4. Before sending the agreement it blocks the other port and once the agreement is sent, a connection is established between switch-3 and switch-4 with properly assigned root and designated ports in forwarding state which is demonstrated in Figure 2.9.

In Figure 2.10, the synchronization for the link between switch-3 and switch-4 stops. The ports on the link between Switch-3 and switch 2 are now set to discarding

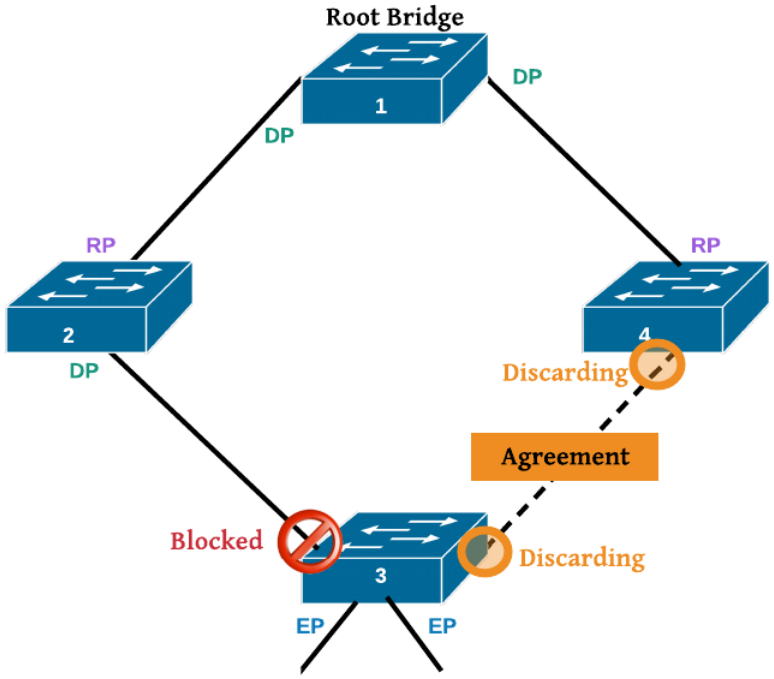

Figure 2.9: STEP-5: RSTP Synchronization Process

and switch-3 sends a proposal to switch-2. Switch-2 already has the best path to the root bridge and its root bridge is already elected, so switch-2 rejects the BPDU and both switch-3 and switch-2 realize that the link between them should be blocked to prevent a loop. Hence the port on switch-3 is set as alternate port and the switch-2's port is set to designated port.

The proposal/agreement and synchronization process stops and there is a new loop free tree formed.

Now that the RSTP synchronization is explained, let us further understand two

vulnerabilities in the modern RSTP approach.

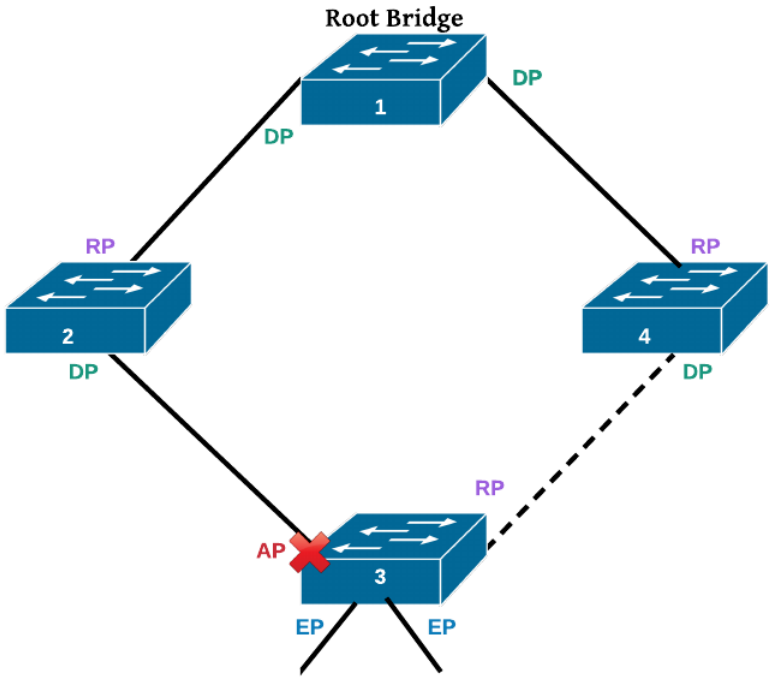

Figure 2.10: STEP-6: RSTP Synchronization Process

1. **Race-condition:**

   Consider a sample ring topology, as seen in Figure 2.11, that is running RSTP in it. Ring topologies frequently prefer RSTP as the protocol of choice and the scenario that is discussed is an actual potential scenario. The ring topology shown in Figure 2.11 is used for demonstration. With RSTP implemented in all the switches of the topology, the entire ring topology is split into two parts to avoid loops. This is achieved by blocking the port on switch-X. A set of

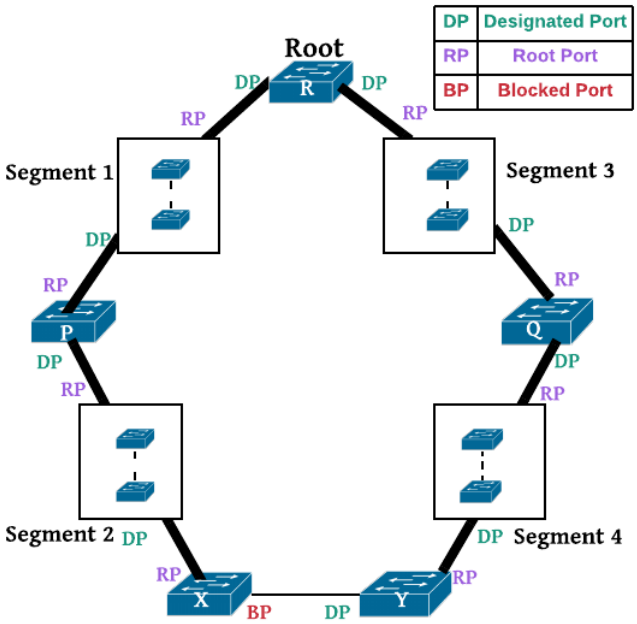

Figure 2.11: Sample ring topology to explain race condition in RSTP

switches that are connected as point-to-point links in a linear fashion are bundled together as segments. For discussion, a couple of switches in between this linear chain of switches are mentioned as switch P and switch Q. Now consider the scenario where the root bridge of this ring topology fails, which is shown in Figure 2.12. The topology is now split into two equal parts. One starting from the left of the root till switch X and the other one starting from the right of the root till switch Y.When each of these segments realize that there is no root, they start to elect their own roots separately by exchanging BPDUs. Assuming that P and Q are the most eligible switches in their respective segments and they are elected as temporary root bridges in their segments. Switch X which has the blocked port and was acting as the separator for both the parts plays

an important role. Information about this new root election will be propagated down the tree and there are two possibilities here.

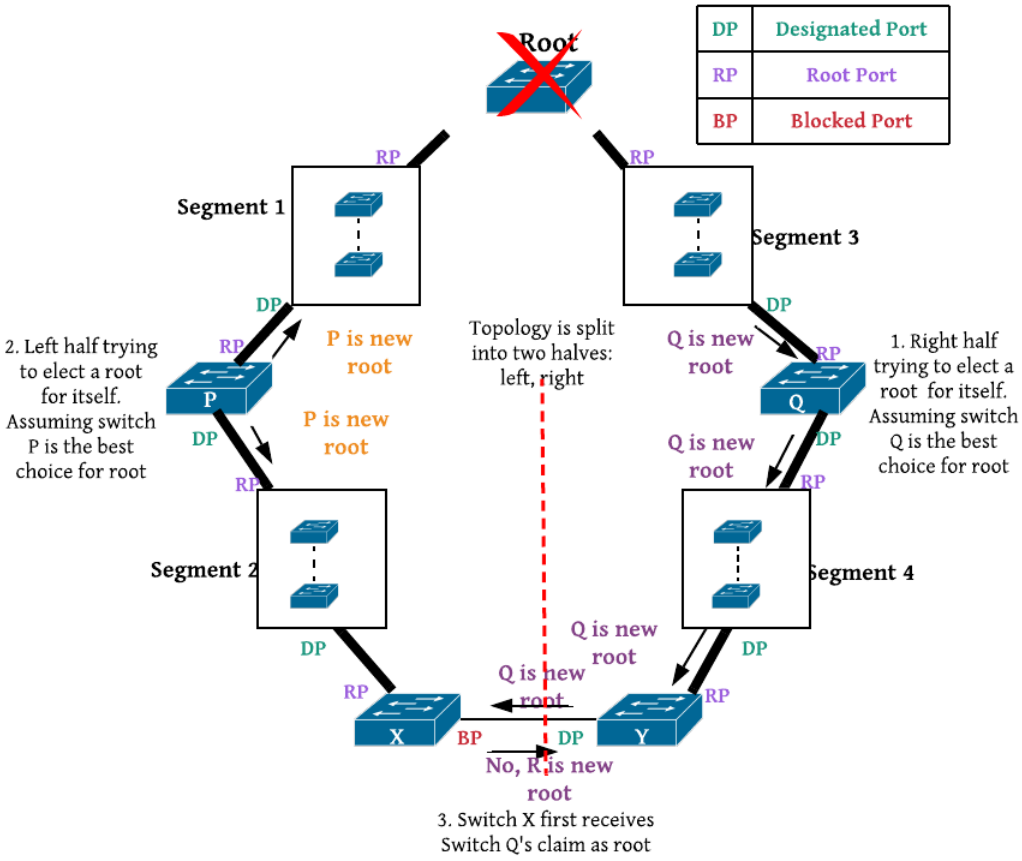

Figure 2.12: Root failure leading to race condition in RSTP

Information about switch Q claiming itself as root reaches switch X first. Information about switch P claiming itself as root reaches switch X first. In the first scenario, if switch X gets the BPDU about switch Q claiming itself as root first, switch X already has a root port and believes that the root R is still up,

so it rejects the proposal message that it gets and proposes that root R is the root bridge to switch Y. Switch Y believes that it can reach the root via switch X and propagates this information further.

Segments 4, switch Q and segment 3 adapt to this information assuming that they can reach the root R via switch X. Figure 2.13 represents this phenomenon.

Later, when switch X receives the information about switch Q claiming itself as the root bridge, the old information about switch R being root is flushed out as this information is received on the root port of switch X. Based on the priorities of switches P,Q, X and Y, a new root bridge is elected and the synchronization is carried out in the rest of the topology. This stale information of the root bridge in switch X might lead to a delay in the election of a new root in RSTP since this stale information is propagated in the entire topology and switches for a brief amount of time believe old root R can still be reached via switch X.

If switch X receives information about switch P claiming as the root first compared to switch Q, this problem is avoided as switch X will immediately flush out its old root information and this information is properly synchronized in the rest of the topology. The additional time to flush the old root information from switches Y and further is not required in this scenario.

Compared to a link failure in a ring topology, the failure of a root bridge results in much higher convergence time. This additional time is attributed to the time it takes for each of the separated segments trying to elect their root and then proceeding to elect a single root for the entire topology by merging

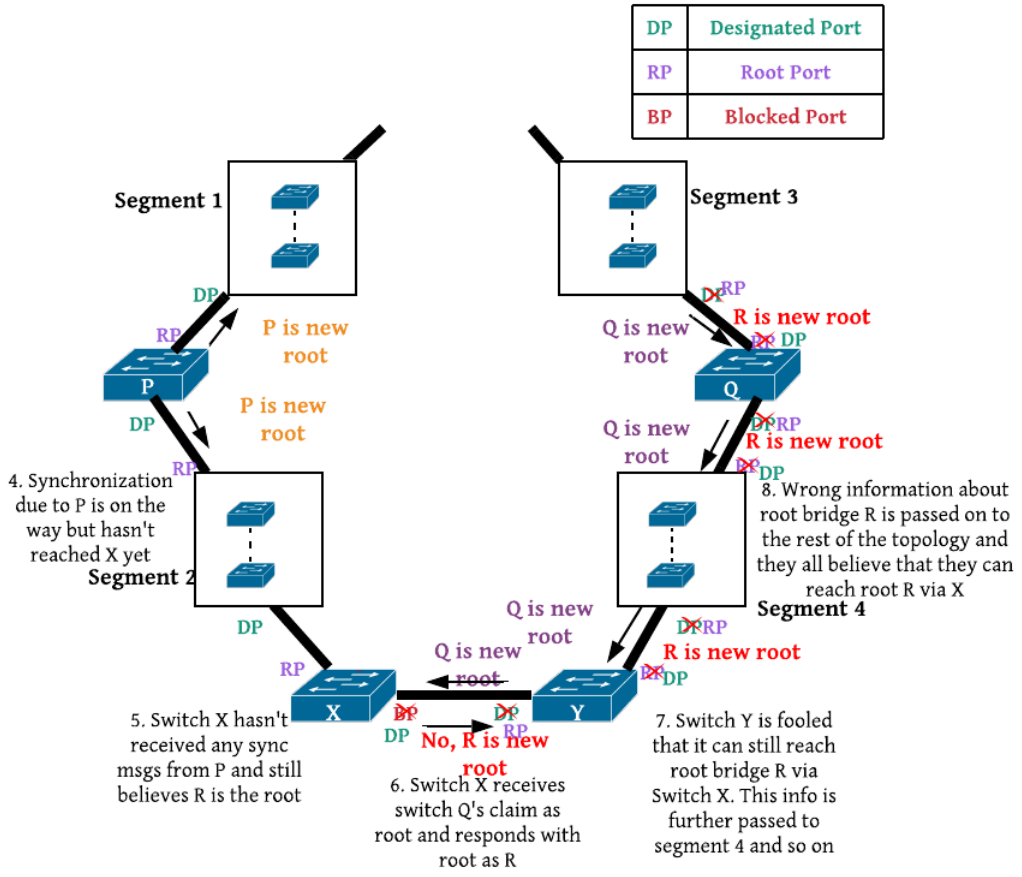

Figure 2.13: Ring topology demonstrating high convergence time due to root failure

them. Also, the convergence time heavily relies on how the information is travelling in the topology. Based on the processing speeds, event sequencing and other configuration in the switches, convergence times may vary for different topologies. This interesting problem in RSTP is referred to as Race condition. As discussed in this scenario, it would take more time for the topology to converge when the root fails and information about switch Q reaches switch X first.

There is another problem in complex ring topologies where RSTP is configured and the root bridge fails . The next section discusses that issue.

2. **RSTP Count to infinity problem:**

   In the earlier section, it is explained how the failure of a root bridge results in a higher convergence time due to the circulation of old root bridge information in the topology even after the failure of the root bridge. This is a result of the caching mechanism that RSTP incorporates. This section discusses an even worse possibility where the stale root bridge information persists in the topology for longer time due to the formation of loops in the topology after the root failure. The following ring topology shown in Figure 2.14 is used to explain the issue. Similar to what was seen in the previous scenario, segments 1 and 2 are a series of switches connected using point-to-point links.

   In scenarios where an entire ring topology is connected to the root bridge via a single link, this issue is more likely to happen. Assuming that the link between R and switch A is broken or the root bridge R is completely down. In either case, it leads to the root bridge R completely being expelled from the remainder of the topology and the network comprises of a physical loop even though they are logically separated by a blocked port at switch C. In the example that is currently being discussed in Figure 2.14, it is assumed that Switch A has the best priority and is the rightful choice for the root bridge.

   Once switch A misses 3 hello's it is convinced that the root is down and claims itself as the new root and sends the corresponding sync messages down to the

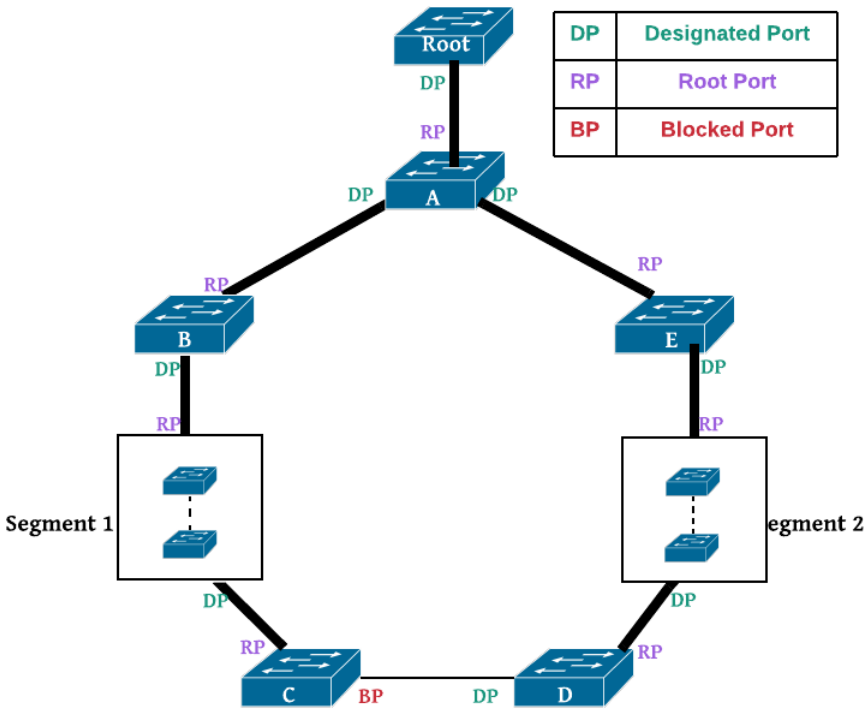

Figure 2.14: Sample ring topology to demonstrate count to infinity problem in RSTP

rest of the topology. These message are propagated from Switch A to Switch B and switch E and these switches carry them further down the topology. The speed at which these messages are propagated down the network depends on factors like processing speed of the switches, event sequencing etc. As discussed in the previous section about the existence of a race condition, there is a high possibility that the BPDU messages carrying the information about switch A as root and via the path of switch E-segment2-switch D reaches switch C first compared to the path via switch B. Switch C learns about switch A claiming

it as the root via the path from switch E, switch D and hence receives it on its blocked port. Since the information is received on its blocked port and it already has root R as its root, this information is considered inferior by switch B and responds to switch C with a proposal that switch R is the root bridge and unblocks its blocked port. Switch D receives this information and believes that root bridge R can still be accessed via switch C and updates its information. This information is further propagated to all the switches in segment 2 and then eventually to switch E and then switch A. The above process is pictorially described in the Figure 2.15

Meanwhile, BPDUs traversing via the path switch B-segment-1 finally reach switch C. Switch C receives this information about the new root

on its root port and hence updates its root bridge information as shown in Figure 2.15. This information is further synchronized to switch D and switches in segment-2. Switch D and all the other switches update the false root information that they previously received with the new correct root information. But the problem here is that the information about the false non-existent root is already propagated further and has reached switch A. Switch A has already updated its root information as root R which doesn't exist in the network anymore. This information is sent down the topology as shown in Figure 2.16.
The new root information is chasing the old stale root information as seen in Figure 2.17.

Since the blocked port is also now unblocked, this process runs in a loop. The stale root information circulates in the topology because of the oscillations

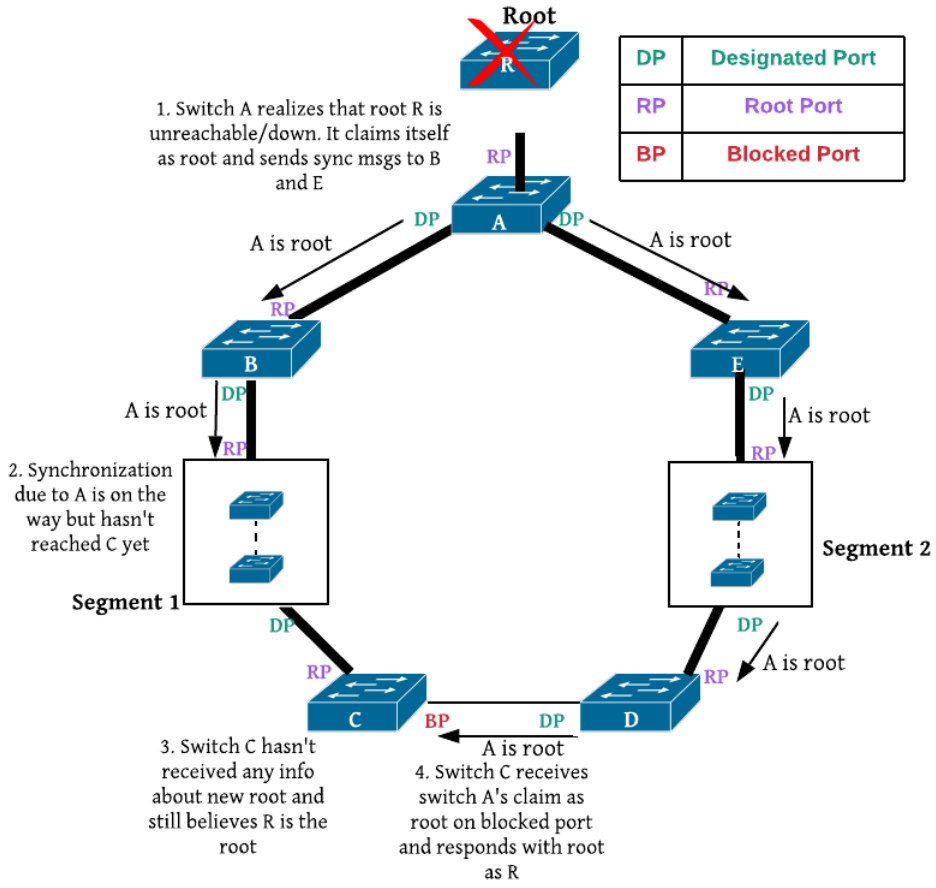

Figure 2.15: Root failure in sample ring topology running RSTP

in the loop and this only ends when the MessageAge timer reaches MaxAge. When the MaxAge is hit, the old stale information is flushed out and the whole network agrees on a common new root. Since the network has to wait for the timer to hit its MaxAge, this problem is named as the count to Infinity problem. Count to infinity is a result of the race condition in RSTP as discussed in the previous section, where the failure of a root results in a physical loop.

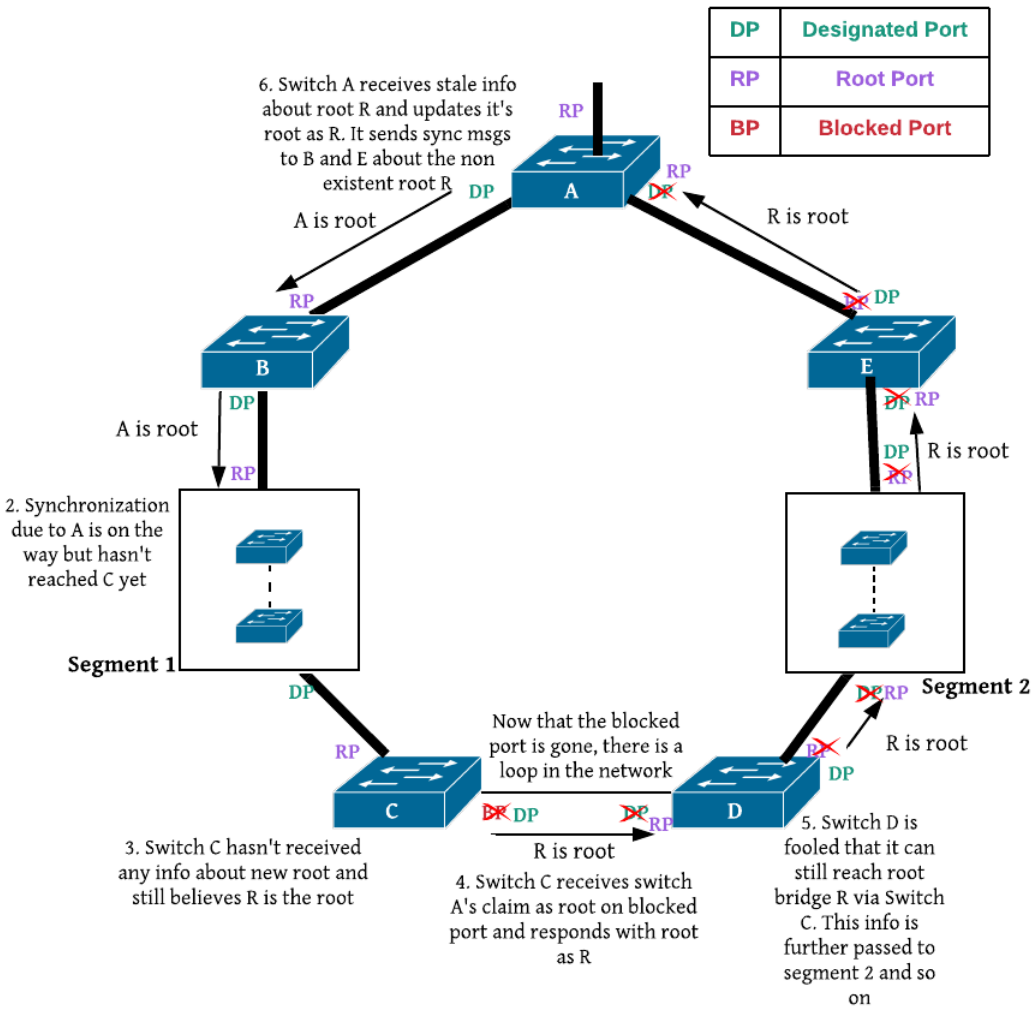

Figure 2.16: Circulation of stale root information in RSTP

Count to infinity problem in RSTP may not be a frequent problem in simple ring or triangle topologies but it poses a serious threat in networks that have multiple redundant links. The topologies and scenarios discussed in the above

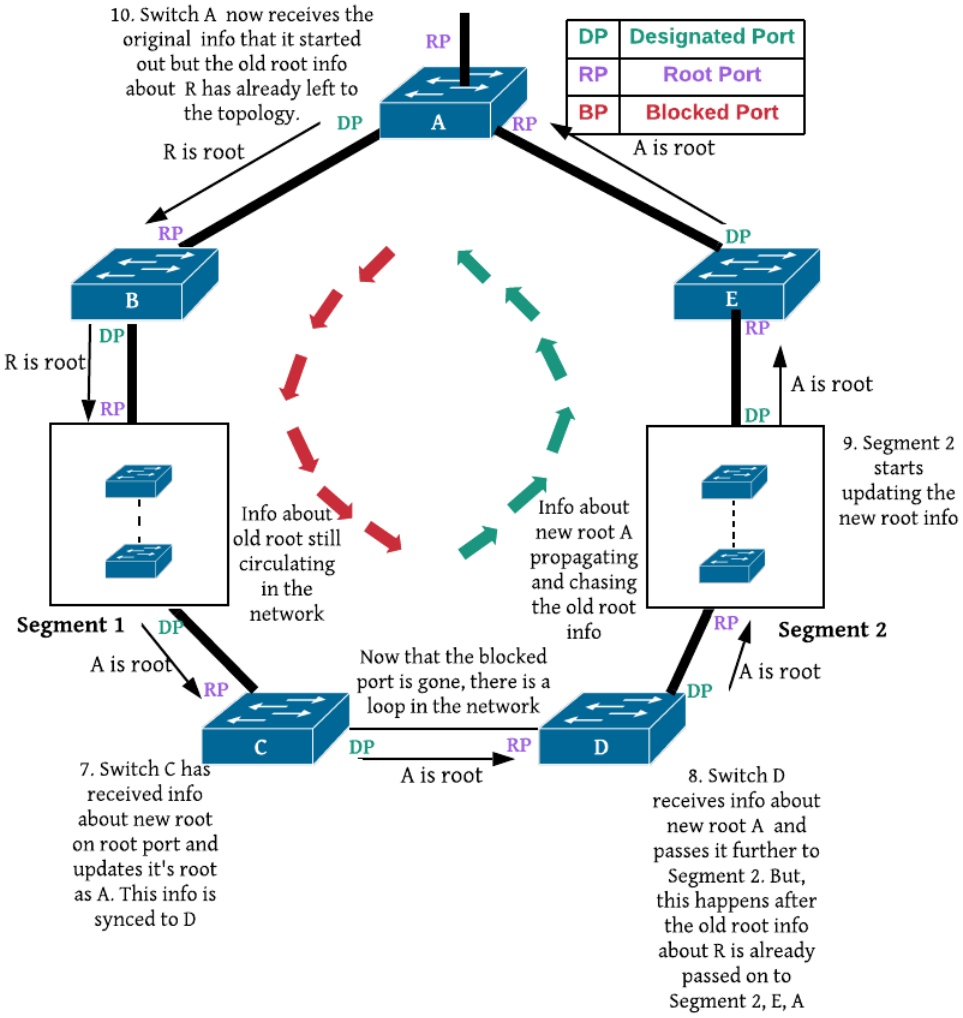

Figure 2.17: New root information chasing the stale old root information leading to the count to infinity problem in RSTP

examples are very likely to occur in many real-time networks with rich set of redundancies. Studies[15][13] reveal that RSTP took upto 30 seconds to converge in some complex topologies.

2.4 Proposed Multi Mesh Tree Protocol

Multi mesh tree protocol is an enhancement to a new protocol that is currently under development as an IEEE standard called the Mesh Tree Protocol(MTP). Mesh Tree Protocol is based on a Mesh Tree Algorithm that offers a completely new approach to solving the problem of loop avoidance in Meshed networks. In current loop-avoidance protocols like STP and RSTP, in order to avoid loops in a meshed network, they construct a single logical tree. At any given time these protocols have a single logical tree and the redundancy in the network is not fully utilized. Whereas, mesh tree protocol builds multiple tree paths from each switch to the root and ensures that there are alternative paths that are ready in case of any failures in the network. Similar to STP and RSTP, root bridge plays the most important role in this protocol as each switch in the network computes a tree to the root. In MTP, a virtual identifier is manually assigned to the root. Each switch acquires VIDs from upstream(closer to root) switch and propagates its VIDs. Downstream switches acquire VIDs from the propagated information and this is propagated across the network and each switch build a set of its identifiers based on which port it receives this information.As we designate a root, in the event this root fails, it is important to address the need for another root to take over. This thesis introduces root redundancy in the Mesh Tree Protocol and is called the Multi Mesh Tree Protocol. Mesh Tree protocol has made significant progress in the recent years and the latest work [27] compares MTP with RSTP and demonstrates MTP's superior performance in minimal processing and extremely reduced recovery time. The working and the algorithm behind the mesh tree protocol is published in various papers and this thesis focuses

mainly on the enhancements over the mesh tree protocol and how root redundancy is introduced. To demonstrate better, it is key to understand the terminologies that are used in the mesh tree protocol. The following section will introduce and describe the key terminologies that are used in the MMTP and also help us in understanding MTP's implementation of root redundancy. Figure 2.18 will be used as an example to explain the terminologies better.

Mesh Tree Protocol Virtual Identifier(VID)

In MTP, multiple trees are constructed using a simple numbering scheme. Each switch in a meshed tree is assigned a set of Virtual identifiers(VID). Each VID for a switch represents the unique path from the switch to the root. A VID is a series of numbers separated by a dot in between and the first number in the series is the VID of the root switch. The numbers after that are the outgoing switch port numbers attached in the path from the root switch to the switch that is receiving the VID. Example: In figure, for switch Node-1 with VID 1.2, 1 is the VID of the root switch and the 2 is the port number of the root switch in the path from root to the current switch.

An example of a 5 node topology after successfully creating its primary and secondary VID tables is shown in the Figure 2.18.

Primary Root

The Primary Root acts as the root for the primary tree that is built as a part of the MMTP implementation. In the 5-node topology in Figure 2.18, Node-0 is the Primary Root.

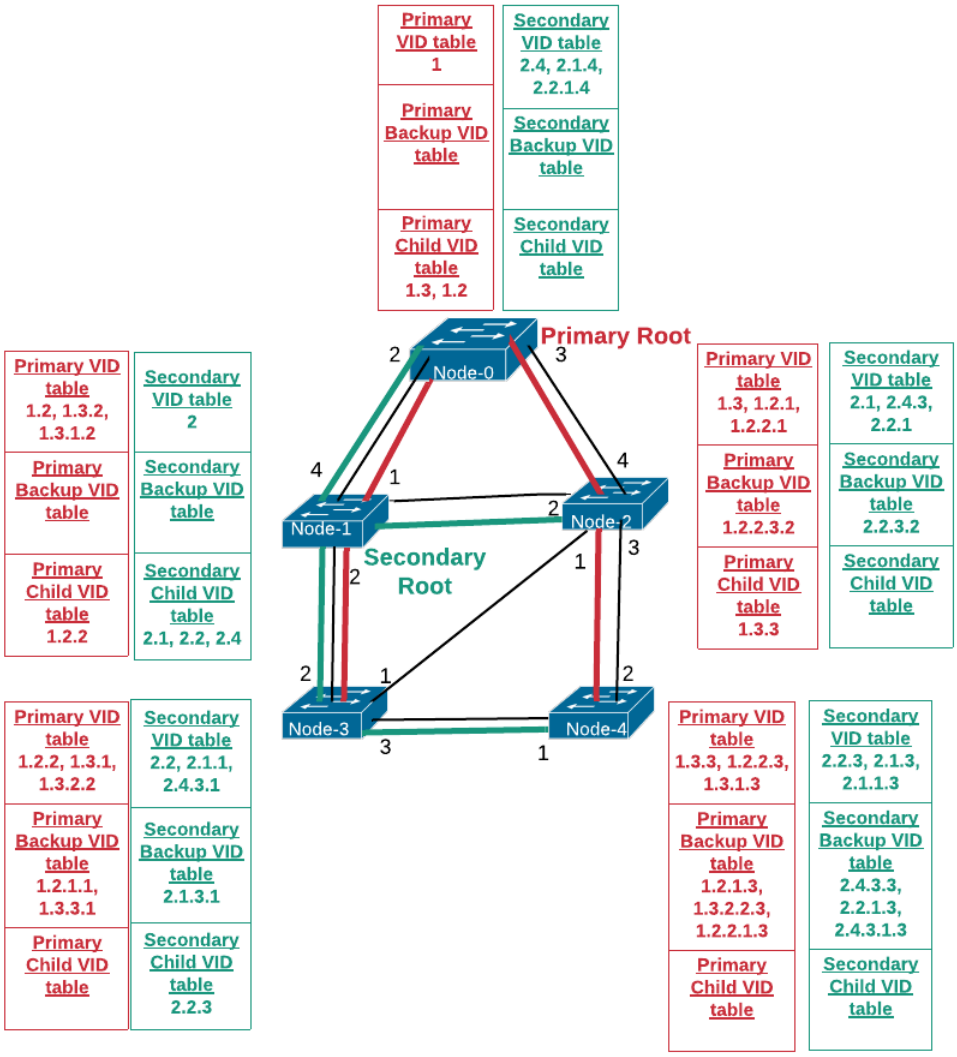

Figure 2.18: A Stable 5 Node Topology running the enhanced MMTP protocol

Primary VID

Each Primary VID for a switch represents a unique path from the switch to the Primary Root. During the initialization of the protocol, the user manually assigns a Primary VID to the Primary Root. In Figure 2.18, Node-0 is the Primary Root and is assigned Primary VID as 1.

Primary VID table

Every switch will have a Primary VID table that contains the set of Primary VIDs that it received. The Primary VID table stores a maximum of 3 Primary VIDs and hence if it receives more than 3 Primary VIDs, it ranks the Primary VIDs based on the efficient path to the Primary Root and chooses 3 best Primary VIDs to store in the Primary VID table. This maximum(currently 3) for the number of VIDs that can be stored in the Primary VID table can be configured based on the network needs.

Primary Backup VID table

Other Primary VIDs remaining after a switch chooses its best 3 Primary VIDs are saved in the Primary Backup VID table.

Primary Child VID table

For a given Primary Tree, the switches also create a parent-child relationship. The Primary VIDs generated for the children by a parent switch are stored in the Primary Child VID table of the parent switch. In Figure 2.18, the tree colored in red represents the Primary Tree and since Node-1 and Node-2 are children of Node-0(Primary Root), the Primary Child VID table of Node-0(Primary Root) has 1.2 and

1.3 which are the primary VIDs generated by the parent switch(Node-0) for Node-1 and Node-2 respectively.

Secondary Root

The Secondary Root acts as the root for the Secondary tree that is built as a part of the MMTP implementation. In the 5-node topology representation in Figure 2.18, Node-1 acts as the Secondary Root.

Secondary VID

Each Secondary VID for a switch represents a unique path from the switch to the Secondary Root. During the initialization of the protocol, the user manually assigns a Secondary VID to the Secondary Root. In Figure 2.18, Node-0 is the Secondary Root and is assigned Secondary VID as 2.

Secondary VID table

Every switch will have a Secondary VID table that contains the set of Secondary VIDs that it received. The Secondary VID table stores a maximum of 3 Secondary VIDs and hence if it receives more than 3 Secondary VIDs, it ranks the Secondary VIDs based on the efficient path to the Secondary Root and chooses 3 best Secondary VIDs to store in the Secondary VID table. This maximum(currently 3) for the number of VIDs that can be stored in the Secondary VID table can be configured based on the network needs.

Secondary Backup VID table

The excess Secondary VIDs remaining after a switch chooses its best 3 Secondary VIDs are saved in the Secondary Backup VID table.

Secondary Child VID table

For a given Secondary Tree, the switches also create a parent-child relationship. The Secondary VIDs generated for the children by a parent switch are stored in the Secondary Child VID table of the parent switch. In figure Figure 2.18, the tree colored in green represents the Secondary Tree and since Node-0, Node-2 and Node-3 are children of Node-1(Secondary Root), the Secondary Child VID table of Node-1(Secondary Root) has 2.1, 2.2 and 2.4 which are the Secondary VIDs generated by the parent switch(Node-1) for Node-0, Node-2 and Node-3 respectively.

Mesh Tree Protocol Data Unit(MTPDU)

MTP uses a special type of message to communicate between the switches called Mesh Tree PDU. Depending on the message that needs to be communicated, the MTPDU is constructed with a set of different fields. The overhead for these MTPDUs is very low.

Types of MTPDUs:

1. **JOIN Message:**

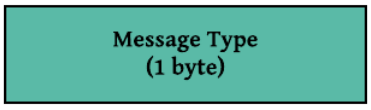

Figure 2.19: Traditional MTP Join Message BPDU

A switch running MTP, that is not a root and doesn't have any VIDs, sends a JOIN message to express its desire to join the mesh tree. The join message

shown in Figure 2.19 is the simplest BPDU consisting of just the message type field. The message type field is just a byte in size.

2. **Advertisement Message:**

   A switch that already acquired atleast one VID and receives a Join message from another switch wishing to be a part of the mesh tree, sends out an advertisement message that will be used by the requesting switch to be a branch of that mesh tree. Advertisement message shown in the Figure 2.20 is the longest of the MTPDUs that are currently defined. A switch advertises its VIDs from its Primary VID table by appending the port number to the VID.

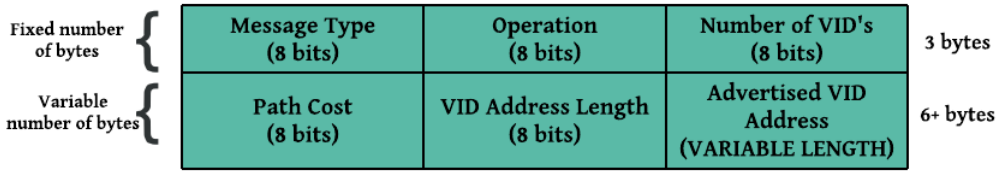

Figure 2.20: Traditional MTP Advertisement Message BPDU

3. **Hello Message:** These messages are exchanged between the switches to ensure health of the switches. Hello messages are sent at regular intervals and can be configured as needed. Hello message shown in Figure 2.21 plays a crucial role in detecting failures in the network.

The MMTP introduces a new root and a totally new tree is built based on the new root. This new root is referred to as Secondary Root and the original root

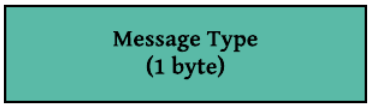

Figure 2.21: Traditional MTP Hello Message BPDU

as the Primary Root. Using the same MTPDUs that are originally used to build the primary tree, the tree number field is introduced in it and accordingly the secondary tree is built in parallel. Now there are two roots Primary Root and Secondary Root and two trees based on them namely Primary Tree and Secondary Tree.

The MTPDUs are now modified to also carry the tree field in them so the switches can differentiate and build the corresponding tree. The following section explains how the new tree number field is introduced in the MTPDU. The tree number field is an integer and hence one byte is assigned for it. This enables the protocol to have more than 2 trees in the future.

The join message has not been changed in this version of MTP or MMTP because any switch that doesn't have any VIDs currently can request to join the both the trees. Irrespective of the tree number field, based on the switch that receives the join message, if it has a Primary VID it sends out its the VID advertisements on tree 1 and if it has Secondary VID it sends out its VID Advertisements on tree 2. Since there was no need for the join message to be treated differently, the author decided to not alter it.

The hello message has been altered and the tree number field is added. This decision was taken because the hello message is used to monitor the health of the

tree and since there are two trees, their health needs to be monitored individually. The original hello message was just a single byte and the latest MMTP version added a tree number field to it. So the new Hello MTPDU has 2 bytes as shown in Figure 2.22.

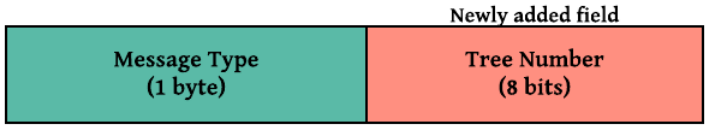

Figure 2.22: New MMTP Hello Message BPDU

The advertisement message was also modified to differentiate the messages based on the tree number. When a VID advertisement is sent, the tree number is added to that message as shown in the Figure 2.23 so as to let the switch know which table to add the VID received.

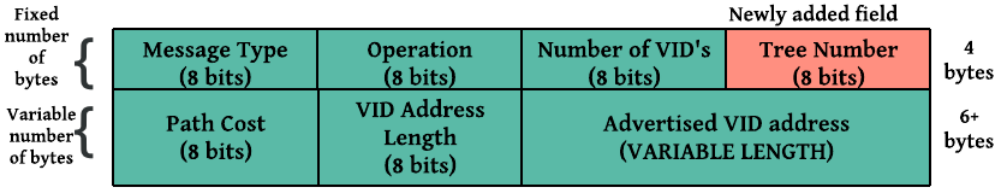

Figure 2.23: New MMTP Advertisement Message BPDU

When MTP is started on the switches, each switch falls into these 3 stages and their behavior is explained in the following sections

Stage-1 Initialization:

1. ***If the switch is a primary root:***

   (a) Add assigned VID to its primary VID table, send out an advertisement to all the switches that are connected on its control interfaces with tree number set as 1.

   (b) Flood Join messages on all its control interfaces as it doesn't have any secondary VID

2. ***If the switch is a secondary root:***

   (a) Add assigned VID to its secondary VID table, send out an advertisement to all the switches that are connected on its control interfaces with tree number set as 2

   (b) Flood Join messages on all its control interfaces as it doesn't have any primary VID

3. ***If the switch is a non-root:***

   (a) Since the non-roots don't have any VIDs, on startup, they flood their control interfaces with join messages

Stage -2 Tree Building:

When a primary root switch receives a join message, as it has a primary VID, it sends out an advertisement with tree number set as 1 on the interface that it received the join message. Since it doesnt have any secondary VID yet, it doesn't send any

advertisement for tree 2. When the primary root receives advertisement on tree2, it adds that VID to its secondary VID table and sends out VID advertisements with tree number set as 2 on its control interfaces.

Initially the secondary root doesn't have any primary VID, so it doesn't send any advertisement for tree 1. Then it checks if there is a secondary VID and since it has one, it sends out an advertisement with tree number set as 2 on the interface that it received the join message. When the secondary root receives a VID advertisement on tree 1, it adds that VID to its primary VID table and sends out VID advertisements with tree number set as 1 on its control interfaces.

Root failure convergence in MTP

After the primary tree is built, a *tree-stable* flag is set to true in each switch and this ensures that the primary tree is up and there is a primary VID in each switch as shown in Figure 2.24. Another field called *tree-to-use* is set as 1 initially when the protocol is using Primary Tree.

Consider the case where the Primary Root of a network that has switches running the Multi Mesh tree protocol fails as shown in Figure 2.25.

The following are the sequence of events that take place:

1. In the Primary Tree, the children of the Primary Root, Node-1 and Node-2 will miss 2 hello messages and quickly realize that their connection to the Primary Root is lost.

2. The children will remove the VIDs received from the primary root and send out VID deleted advertisements to its children.

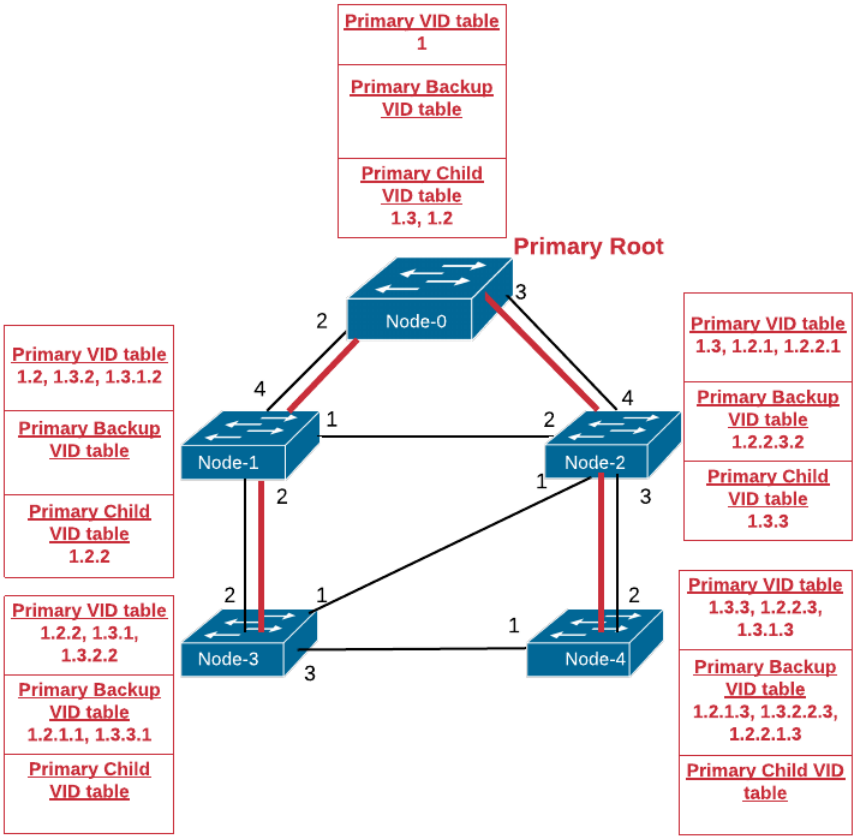

Figure 2.24: 5 Node - Primary Tree and VID Tables

3. Node-1 detects that its connection to Primary Root is lost and removes the VID 1.2 that it got from Primary Root. This deleted VID advertisement is sent to Node-2, and Node-3. Similarly Node-2 also detects that its connection to Primary Root is lost and removes the VID 1.3 that it got from Primary Root. This deleted VID advertisement is sent to Node-1, Node-3 and Node-4 as shown in Figure 2.26.

4. Node-1 deletes 1.3.2 as it got VID delete advertisement from Node-2. This

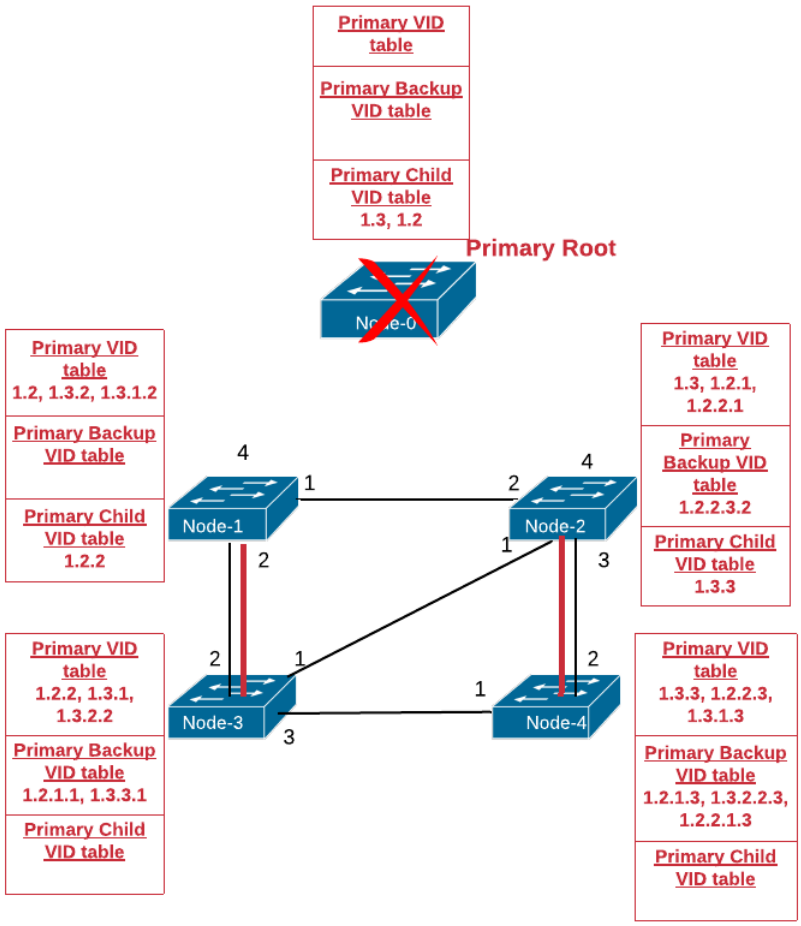

Figure 2.25: 5 Node - Primary Tree - Stage 1 after Root Failure

deleted VID advertisement is sent to Node-2, and Node-3. Node-2 detects that its connection to Primary Root is lost and removes the VID 1.3 that it got from Primary Root. This deleted VID advertisement is sent to Node-1, Node-3 and Node-4.

5. Node-3 deletes 1.2.2 and 1.3.1 as it received VID delete advertisements from

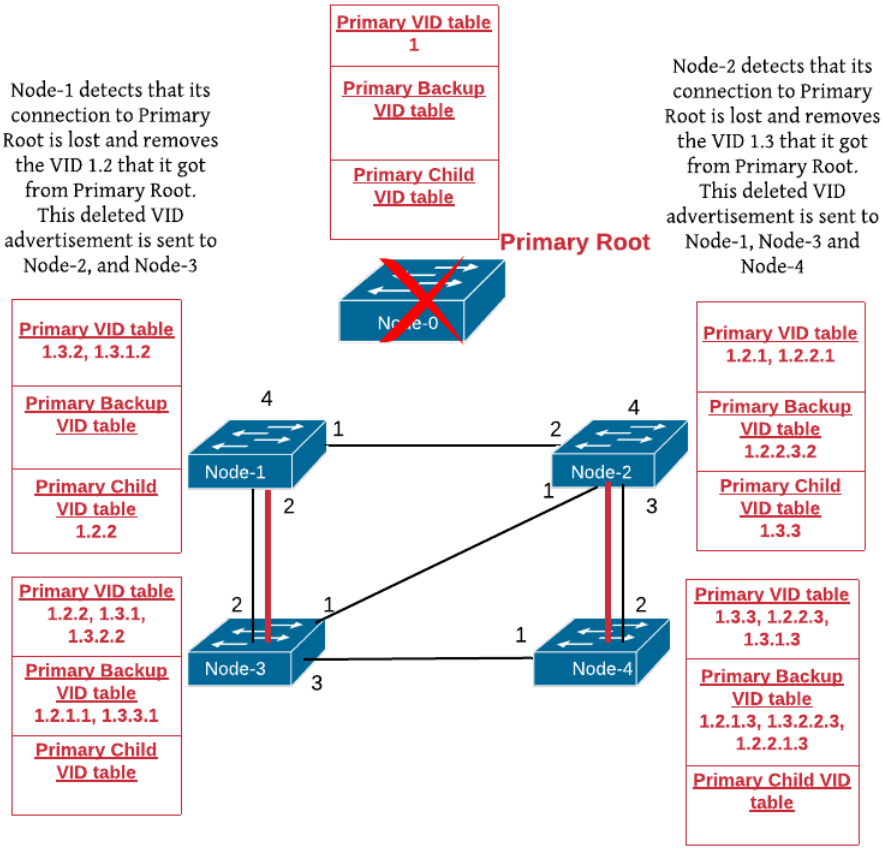

Figure 2.26: 5 Node - Primary Tree - Stage 2 after Root Failure

Node-2 and Node-1. These deleted VID advertisement are sent to on its control ports except the one from which it received the advertisement message

6. Node-4 deletes 1.3.3 as it got VID delete advertisements from Node-2. The deleted VID advertisement is sent to Node-3 as shown in Figure 2.27

7. The VID delete advertisements are further propagated and every switch will remove Primary VIDs derived from the deleted VIDs and finally end up with

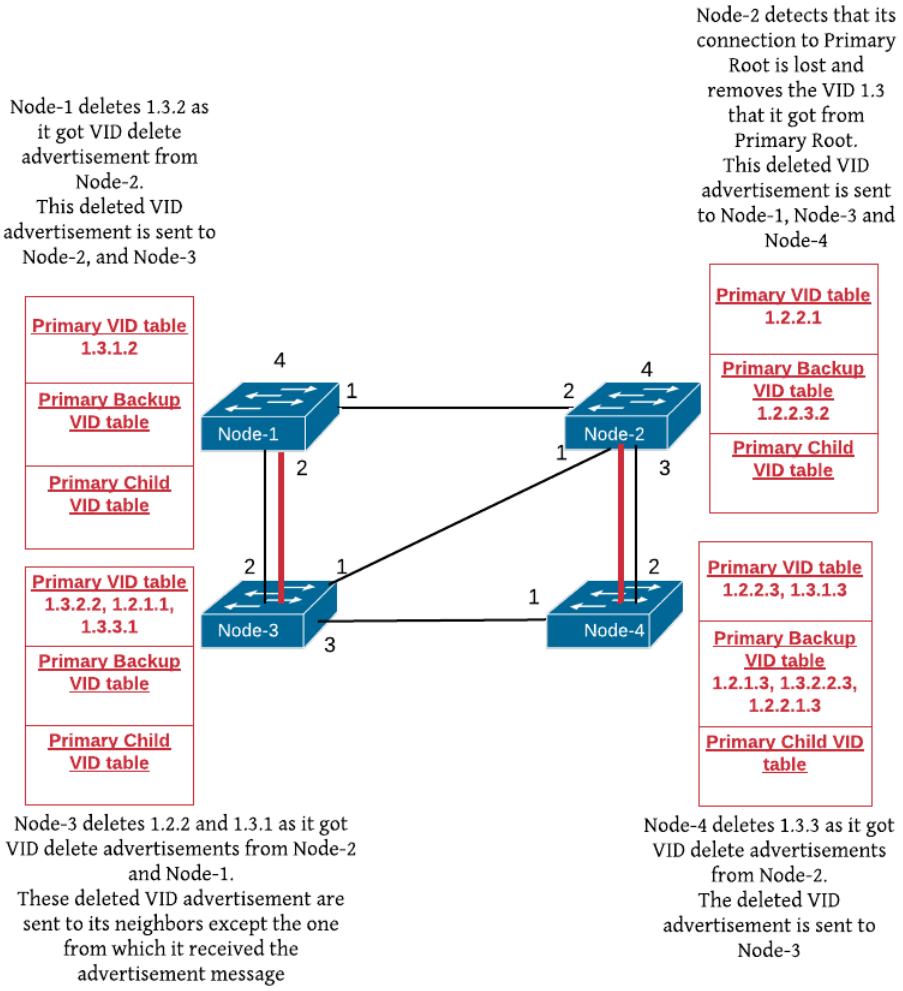

Figure 2.27: 5 Node - Primary Tree - Stage 3 after Root Failure

no Primary VIDs in the Primary VID table as shown in Figure 2.28.

8. Once a switch checks its primary VID table and realizes that it is empty, it immediately sets the *tree_to_use* flag to 2 and logs the time at which it is switching to the Secondary Tree.

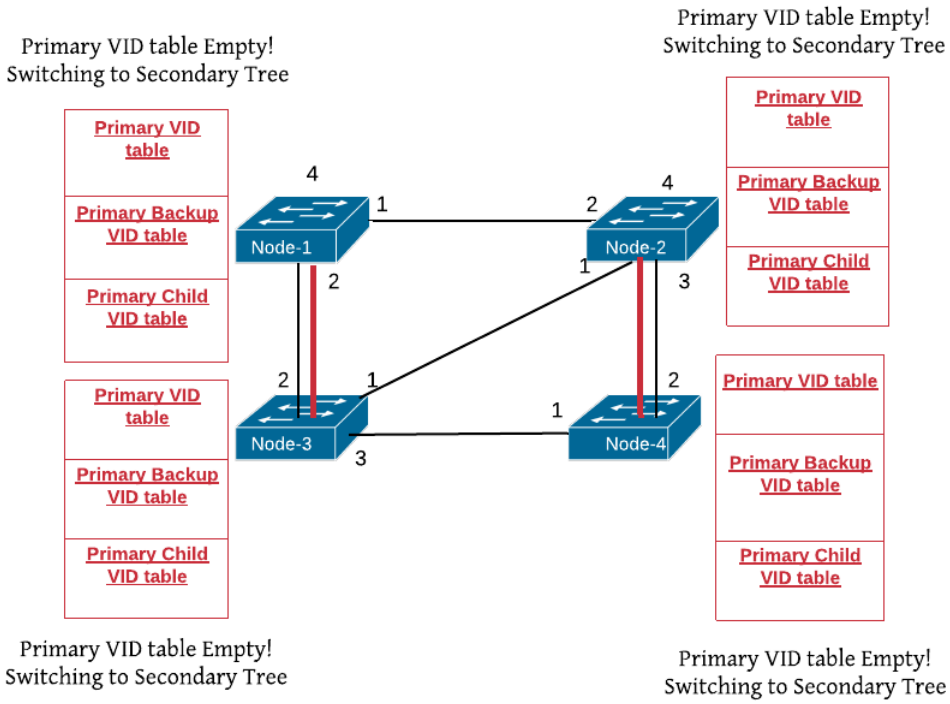

Figure 2.28: 5 Node - Primary Tree - Final Stage after Root Failure

9. There is a Secondary Tree that is already built and is ready for a fail over. Since Node-0 is no longer available in the network, the Secondary Tree undergoes the following changes.

10. Node-1(Secondary Root) will lose connection to Node-0, so it deletes 2.4 as its child. Node-2 will delete 2.4.3 from its VID table. This deleted VID advertisement is sent to Node-3 and Node-4 as shown in the Figure 2.29

11. Node-3 and Node-4 will delete the Secondary VIDs starting with 2.4 that were generated due to Node-0. The final Secondary Tree that will be used for frame forwarding looks like Figure 2.30

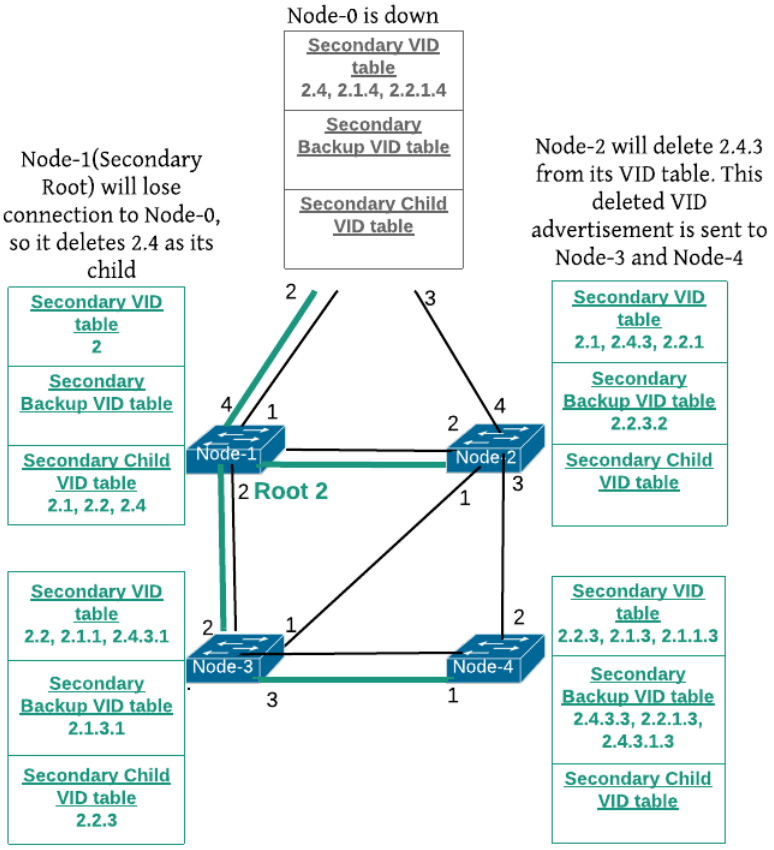

Figure 2.29: 5 Node - Secondary Tree after Root Failure

Thus, the protocol is able to do a fail-over to a secondary tree quickly with minimal delay. The only delay is for the switches to realize that the primary root is down and the time it takes to remove the primary VIDs from its primary tables. The fail-over introduces very low delay and hence the protocol is able to achieve significant improvement in the convergence time compared to the traditional RSTP protocol where there is another additional delay for root election and tree building.

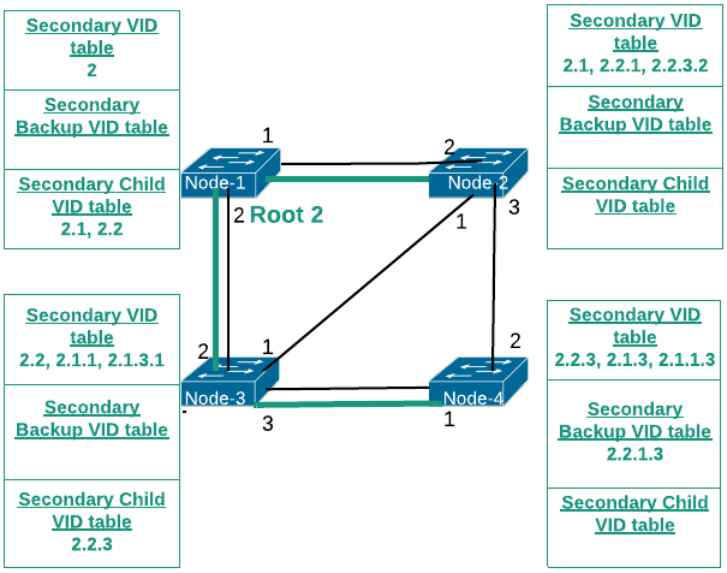

Figure 2.30: 5 Node - Final Secondary Tree after Root Failure

Chapter 3

Methodology

3.1 Implementation of the Protocol:

The implementation of MTP was done as a software written in C language and root redundancy is introduced in the Mesh Tree Protocol by modifying the latest version of the MTP[29] code that was written at the time this thesis was done. The testing and collection of results was done using an automation suite that was written in Python. The hardware for the implementation of this enhancement for the Mesh Tree Protocol was acquired using GENI(Global Environment for Network Innovations). GENI[17] is an open infrastructure for at-scale networking and distributed systems research and education that spans the US. It provides the infrastructure needed to carry out networking research. In this thesis, a set of compute resources like switches, network links to connect them as desired were acquired from GENI and configured according to our needs. The latest version of the MTP was implemented using C code that is supposed to be running in a Unix-like environment based on the Linux kernel. Hence, we used compute resources from GENI that are Linux distributions. MTP is then run as a software in the user-space of the operating system on the reserved compute resources/switches in our case. We leverage the advantage of underlying Linux kernel networking TCP/IP software stack and also the the creation of raw Ethernet II frames to send the Protocol Data Units. The author used three

different topologies to implement, test and collect the performance metrics. These 3 topologies are mirrored for RSTP performance collection too. The topologies are shown in the later sections of this thesis.

3.2 Setup and Testing of the Protocol:

The latest version of the MTP code that is currently being worked on by Peter Willis[29] is downloaded from the GIT repository and the code changes as discussed in the earlier sections were introduced. This code was pushed to all the switches that are part of the topologies created in GENI. For each topology, a python automation script will copy the code to each of the switches in the topology. The input to the automation script is a Resource Specification(RSPEC) file that consists of all the information needed to Secure Shell(SSH) to each of the resources in that topology. The automation script uses a special python library called Paramiko. Paramiko[8] is the python implementation of the SSH protocol and helps us in doing the SSH to each resource and copying the code from the user's machine to the resource hosted on GENI. Once the code is copied and residing on each of the resource in the topology, the protocol is started. Another automation script takes in the information about which nodes you want to assign as primary root and secondary root. This script will prompt the user to enter the time at which the user wants to start the protocol on the three categories of switches namely the primary root, secondary root and the non root switches. The automation script waits till the time that is given in the previous step and starts executing the protocol in the switches. There is a logging system in place that logs every event that takes place in the switches like the arrival of

messages, departure of messages, changes that are made due to message exchanges. This logging system is the key in collecting and calculating the performance metrics for the protocol. Since we have two different trees running in parallel after the introduction of root redundancy, the logging system is also isolated to have two separate log files. This decision was taken to keep the log collection for each of the two trees distinct and to avoid confusion.

The MTP start automation script logs into each of the switches, compiles the C code and uses the GNU screen software[9] in the Linux environment to run the MTP protocol as a process in its own environment. This allows us to collect the output of the protocol execution that has the MTP tables information and prints them into files. These files can be collected later to look at the tables and analyze the protocol behavior. This protocol output that consists of MTP tables data is different from the logging mechanism that is implemented inside the MTP code. The logging mechanism will capture all the communication that is happening between the switches and the printouts of the protocol captures the VID table transitions as a result of these message exchanges.

The automation scripts are uploaded to GitHub[6]. A detailed description of the automation scripts and their working is explained in the following section 3.2.1.

3.2.1 Automation Scripts and their description

MTPConfig.py:

This script installs necessary dependencies for MMTP testing on the GENI nodes and helps in getting a GENI slice ready. It updates the current packages in

the GENI nodes and installs the following softwares on all GENI nodes

- Python Pip
- Chrony for precise time calculation
- TShark[25] for packet capture
- Scapy[24] for raw packet creation and collection

MTPStart.py:

This is the script that transfers all of the MMTP implementation code as well as support scripts and then allows the MMTP implementation to begin running on the node. When it is first started, a prompt will appear with three options: transfer the MTP implementation source code to the bridge nodes (1), transfer the Traffic Generator code to the client nodes (2), or start the MTP implementation on the GENI nodes (3). The script detects if a node is end-node by checking if its name starts with end-node. For end-nodes, the script transfers the traffic generation code that will help in collecting the frame loss metric.

MTPStop.py:

This script simply connects to each GENI node and runs a screen command (screen -X quit) to stop the screen, thus stopping the MMTP implementation on all the GENI nodes.

MTPConvergence.py:

This script does either of the following jobs based on the user's response to a prompt that says Collection(1) — Convergence(2). The procedure is to select Collection first and then run the script again .

- Collection: If the user selects (1), the script collects all the logs from the remote machine and copies it to the local machine from where the automation scripts are executed.
- Convergence: If the user selects (2), the script scrapes all the logs that are collected in the previous step, identifies the important events in the logs and their time stamps, calculates the metrics as explained in the next section 3.3 and puts all this information into an excel file.

3.3 Metrics and Analysis

For this thesis, the testing and analysis of the MMTP protocol was conducted on real hardware. 3 topologies each for MTP and RSTP that were previously used in MTP studies were created in GENI. The latest study by Peter [29] on comparing MTP with RSTP had the same topologies but the experiments were conducted by breaking links at different places in the network. In this thesis, we focus only on one scenario where the root fails and compare the metrics with RSTP under the same scenario.

The primary goal of this thesis is to make sure that the enhanced version of the Mesh Tree Protocol is initialized correctly and is stable. The aim is to get the switches running the MMTP protocol with both the primary tree and secondary tree

built with correct primary and secondary VID tables. An example of how a 5-node topology that is initialized with the MMTP looks after all switches are stabilized with both primary and secondary VID tables is shown in the Figure 2.18 in previous section. For bigger topologies like the 10-node and 17-node, automation scripts were used and the screen log that contains the output of the protocol is collected that printed out the VID tables.

Another major change that was addressed in this thesis in comparison to the previous studies [27][29] is that the Hello timer of the MMTP was cut down to 0.5 seconds from 2 seconds. This decision was taken to reduce the failure detection time. From the results achieved in previous studies [27] and [29] where the hello timer was set to 2 seconds, it is observed that the major contribution to the total convergence time was the detection time. Instead of waiting for 2 hello messages each of 2 seconds, in this thesis the switches wait for 2 missed hello messages each of 0.5 seconds to detect a failure. This experiment gave favourable results and achieved much better results compared to the previous studies[20][27][29][28][12].

The metrics that we use to compare MMTP and RSTP are categorized into two main types. The following section 3.3.1 briefly explains each of the metrics that are collected and how that helps in assessing the superiority of one protocol over other.

3.3.1 Convergence Time

Convergence time is a measure of how quickly a protocol reacts to topology changes and reaches a stable state. To be precise, it is the time taken from the

instance when there was a topology change to the time when the last change in the topology before the topology reaches a stable state. Reaching a stable state or otherwise called as reaching a re-converged state implies that the network running the current protocol is fully available to any clients that are connected to the network and the frames are forwarded correctly. Convergence time is an important metric to assess a protocol's performance.

3.3.1.1 How convergence time is calculated for MMTP

Let the time at which the Primary Root is brought down be called as the *root failure time* or t1. The time at which the neighbor of the Primary Root realizes that it can no longer reach the root be called as t2. The difference between t2 and t1 is termed as *detection time*. The neighbors of the root delete their primary VIDs from their primary table and propagate this information down to the rest of the topology. The neighbor then realizes that there is no primary VID and immediately switches to the secondary tree. This process is continued in the rest of the topology and let the last node in the topology to switch to the secondary tree be called t3. The time difference between t3 and t2 is called the re-convergence time. In this thesis we compare the difference between the occurrence of primary root failure(t1) and the last instance of tree switch(t3). Unlike the RSTP, here MMTP does not have any root election.

Convergence time = Detection time + Re-convergence time

3.3.1.2 How convergence time is calculated for RSTP

The concept of detection time is the same in RSTP too, but there is an addition delay for the root election and once the root is elected, this information has to be synced with the rest of the topology. Hence for RSTP,

Convergence time = Detection time + Root Election Time + Re-convergence /Synchronization time

For each test, the root is brought down and after a few minutes, the logs containing all the events and their times are collected to our local machine. The important times mentioned above are differentiated from the others using a script and are stored in a final document. The same convergence script calculates the convergence time using the equation described before.

3.3.2 Loss of Frames

When there is a network disruption in a topology, during the process of convergence, frames that are in transit over the network get lost. The end to end communication between the source and destination clients is seriously hampered as there will be data loss due to the network disruption. The way this metric is calculated for comparison between MMTP and RSTP is same. Traffic is generated at one end-node or client of the topology. For each of the remaining end-nodes, the number of frames received is taken and thus we determine the loss at each of the other end-node in the topology.

3.3.2.1 Loss of Frames metric in networks running MMTP and RSTP

The topologies that were chosen for analysis in this thesis had end-nodes included for the purpose of collecting the frames lost and comparing MMTP performance with RSTP. When the protocol setup automation script is initialized, it logs into each node and based on the node name, it identifies if a node is end-node and copies a traffic generator code.

The traffic generator code takes in the input of the number of frames and the source. Each frame is constructed with an Ethernet II header with a custom payload that helps us in the analysis. This custom payload is made up of three parts: the source MAC address of the traffic, a sequence number, and filler. Sequence numbering starts at 1 and ends at 1000, with each frame representing one number. The sequence number makes each frame unique and also helps us detect if the frames are lost and also to check if the frames arrived at the destination in the same sequence or not. The sequence number also helps us in identifying any duplicates, if the frames are circulated in the topology for a longer duration. To achieve precise time calculations, Chrony, a Network Time Protocol(NTP) replacement was installed and configured on the nodes in the topology. The suite of Google time servers was used as the official time servers for all of the nodes, and each node had its clocks correctly configured within a millisecond.

This experiment was carried out for three iterations each for MMTP and RSTP. In each of the iteration, the source of the traffic generation was fixed as endnode-2 and at each of the endnodes that are not the source, receive these frames and were captured in a pcap format file. A log file is then generated capturing the

sequence numbers. These logs are finally collected in the local machine using the same automation script, that analyzes the information in the log files and generates the final results of the loss of frames. This analysis is not confined to just counting the number of lost frames but does some additional analysis like the number of duplicated frames, the order in which the frames arrived in case of the root failure.

Chapter 4

Results and Discussion

The following sections showcase the results collected for the 3 topologies as discussed in the earlier sections. All the result collection mentioned below are carried out after the protocols are successfully started and running in all the switches. In the case of MMTP, this meant transferring the C code to each switching node, and the traffic generation script to each client and verifying that the Primary and Secondary Trees are built. For RSTP, this included the installation and configuration of OvS[14], the designation of a root bridge, and any modifications of the resulting spanning tree topology by tweaking Remote Procedure Call(RPC) values for interfaces that needed to be an active, forwarding interface in the spanning tree. After this is achieved, the root bridge is brought down by pulling all the interfaces on the switch down at the same time.

The results in the following sections are represented in the order shown below. For each topology: 1. The successful formation of the Primary VID Tables and Secondary VID Tables existing in parallel (Subsections 4.1.1, 4.3). 2. Three experiments to compare the convergence times (Subsections 4.1.1.1, 4.2.1, 4.3.1) and 3. Loss of frames (Subsections 4.1.2, 4.2.2, 4.3.2) in the event of a root failure and 4. Discussion on the collected results (Subsections 4.1.3, 4.2.3, 4.3.3)

4.1 5 Node Topology Results

4.1.1 5-Node: Primary and Secondary Tree formations

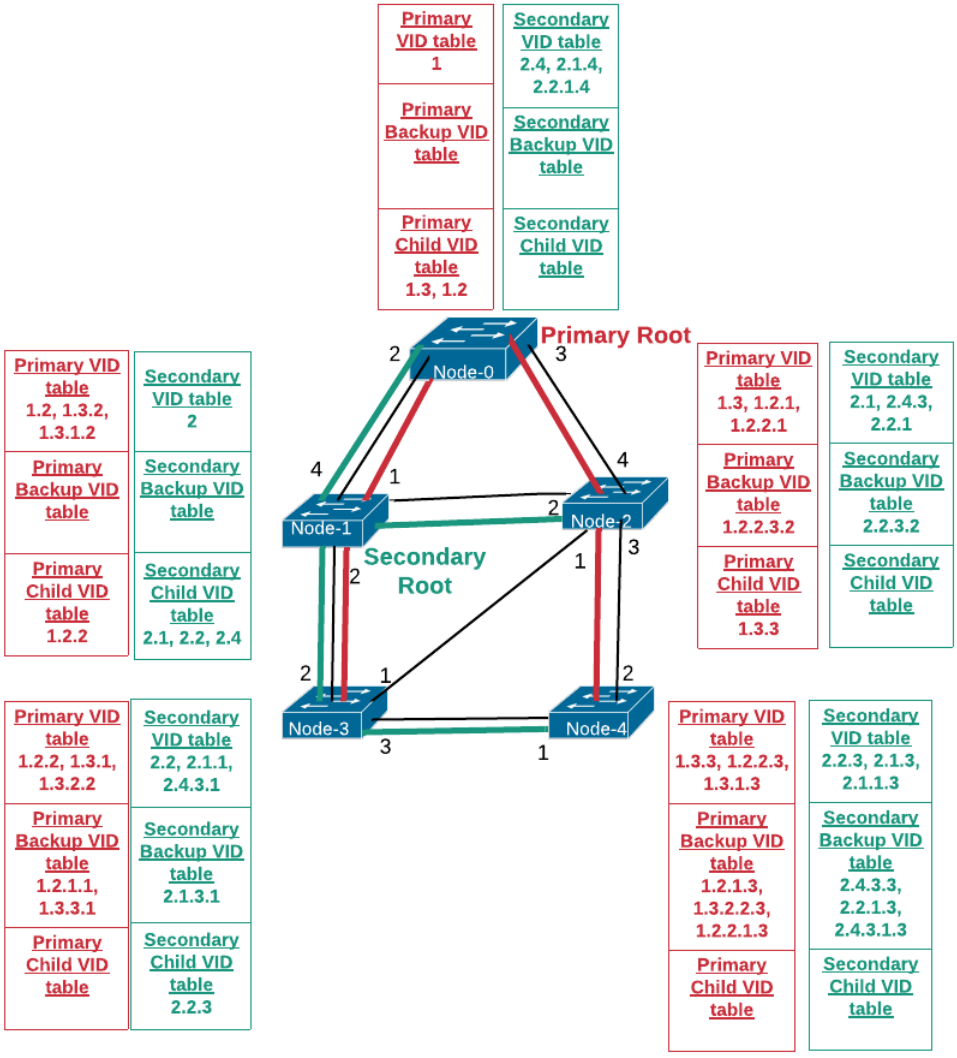

Figure 4.1: 5 Node - Formation of primary and secondary tree tables

4.1.1.1 5-Node Convergence time comparison between MMTP vs RSTP

Table 4.1: 5-Node Convergence times comparison MMTP vs RSTP

Test#	Type of Delay	MMTP	RSTP
Test 1	Detection Time	0.733	5.37
	Root Election Time	0	4.555
	Re-convergence Time	0.007	1.39
	Total Convergence	0.74	11.315
Test 2	Detection Time	0.712	4.645
	Root Election Time	0	2.566
	Re-convergence Time	0.028	1.514
	Total Convergence	0.74	8.725
Test 3	Detection Time	0.749	3.453
	Root Election Time	0	1.536
	Re-convergence Time	0.016	3.037
	Total Convergence	0.765	8.02
Average	Detection Time	0.731	4.489
	Root Election Time	0	2.886
	Re-convergence Time	0.017	1.980
	Total Convergence	**0.74833**	**9.353**

4.1.2 5-Node Loss of frames comparison between RSTP vs MMTP

Table 4.2: End Node 2 sending 1000 frames

Test#	End Node 2 sending 1000 frames	End Node 1	End Node 3	End Node 4
Test 1	MMTP	94	94	0
	RSTP	497	497	1
Test 2	MMTP	72	72	0
	RSTP	491	491	0
Test 3	MMTP	147	147	0
	RSTP	456	456	1
Average	**MMTP**	**104.33**	**104.33**	**0.00**
	RSTP	**481.33**	**481.33**	**0.67**

4.1.3 5-Node Results Discussion

As shown in Figure 4.2, the 5-Node topology is the smallest and the least complex topology among the set of topologies used in this thesis study. Given the less complicated nature of the topology, it was easy to understand and verify the implementation of the protocol on this topology. Node-0 is the Primary Root, and Node-1 is the Secondary Root. The existence of both Primary Tree and Secondary Tree after the protocol is initialized in the topology is shown in Figure 4.1. Each switch is connected to an end-node resulting in a total of 5 end-nodes. MMTP is implemented with a hello timer of 0.5 seconds, and two missed hellos is an indication of a link or switch failure. Whereas in RSTP, the hello timer is 2 seconds, and

three missed hellos indicate a link or switch failure. When Node-0 is disconnected from the topology by bringing the interfaces on Node-0 down, its neighbors Node-1 and Node-2 detect that the connection to Node-0 is down after missing two hellos. Therefore it is observed that the detection time for MMTP is always between 0.5 seconds to 1 second. On average, the detection time for MMTP was 0.69 seconds. Similarly, for RSTP, the detection time is always between 4 seconds to 6 seconds, as

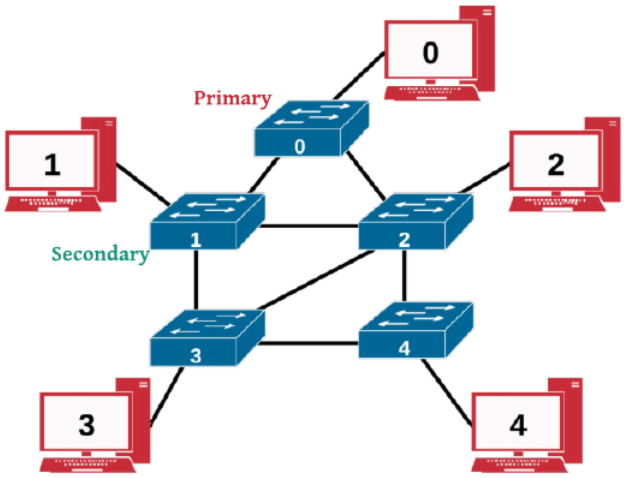

Figure 4.2: 5 Node Topology used to illustrate frame loss

the neighbors have to miss three hellos each of 2 seconds. On average, RSTP took 4.13 seconds to detect a root failure. RSTP showed inconsistent times for the root election time, as highlighted in the Table 4.1. The average root election time for a simple 5-node topology as shown in Table 4.1 is around 2.8 seconds. As discussed in earlier sections, RSTP takes a much higher convergence time when the root bridge is down, and the results validate the same. In MMTP, after the neighbors detect that there is a Primary root failure, they exchange a series of VID delete advertisements

and quickly reach a state where all the Primary VIDs in the Primary VID table are deleted. At this stage, the switch immediately falls back to the Secondary Tree. This process of switching to the Secondary Tree after the nodes detected a root failure is observed to be very less for MMTP. For RSTP, even after the root is elected, it took an average of almost 2 seconds to synchronize the new root information to all the switches in the network.

During the convergence process, frames sent from end-node-2 in the Figure 4.2 get lost in the network. For MMTP, during the time when Node-2 is trying to reach Node-0, frames are lost. Also, end-node-3 loses some frames during this detection time as frames need to travel via the path through Node-0. Since end-node-4 is directly connected to Node-2, the frames that are directed towards end-node-4 are not lost. The results show that, on average, 104.33 frames are lost for end-node-1 and end-node-3, and no frame loss for end-node-4. RSTP convergence took a long time because of the additional root election time and the synchronization time, and hence we see a lot of frames lost compared to MMTP. It is also observed that the frames lost for RSTP depend on the convergence time. In test-1, as shown in Table 4.2, RSTP took 11 seconds to converge and lost 497 frames, whereas, in test-3, it converged faster in 8.02 seconds and lost only 456 frames. Since this is a simple 5 Node topology, the effect on the frame loss couldn't be completely assessed. This metric is further explored in bigger topologies in the next results.

4.2 10 Node Topology Results

4.2.1 10-Node Convergence time comparison between RSTP vs MMTP

Table 4.3: 10-Node Convergence times(in seconds) comparison MMTP vs RSTP

Test#	Type of Delay	MMTP	RSTP
Test 1	Detection Time	0.581	4.379
	Root Election Time	0	5.396
	Re-convergence Time	0.043	3.008
	Total Convergence	0.624	12.783
Test 2	Detection Time	0.732	4.192
	Root Election Time	0	12.207
	Re-convergence Time	0.048	3.028
	Total Convergence	0.78	19.427
Test 3	Detection Time	0.769	3.832
	Root Election Time	0	1.022
	Re-convergence Time	0.026	3.081
	Total Convergence	0.795	7.935
Average	Detection Time	0.694	4.134
	Root Election Time	0	6.208
	Re-convergence Time	0.039	3.039
	Total Convergence	**0.733**	**13.382**

4.2.2 10-Node Loss of frames comparison between RSTP vs MMTP

Table 4.4: End Node 2 sending 1000 frames

Test#	End Node 2	End Node 1	End Node 3	End Node 4	End Node 5	End Node 6	End Node 7	End Node 8	End Node 9
Test 1	MMTP	109	109	0	109	0	109	0	109
	RSTP	693	696	1	745	211	698	212	214
Test 2	MMTP	148	148	0	148	0	148	0	0
	RSTP	696	786	354	847	417	849	417	442
Test 3	MMTP	197	197	0	197	0	197	5	96
	RSTP	381	505	0	503	0	503	0	68
Average	**MMTP**	**151.3**	**151.3**	**0**	**151.3**	**0**	**151.3**	**1.6**	**68.3**
	RSTP	**590**	**662.3**	**118.3**	**698.3**	**209.3**	**683.3**	**209.6**	**241.3**

4.2.3 10 Node Results Discussion

The 10-Node topology as shown in Figure 4.3 has more number of switches, end-nodes and links compared to the previous topology. This introduced some complexity into the network. By growing the topology, some results stayed consistent with the 5 node topology like the detection times for both the protocols don't depend on the size and complexity of the topology but rather depend on the hello timer configurations. Therefore, the detection time averages are similar to what were achieved for the 5 topology. For MMTP, after the root failure detection, the protocol undergoes a process where the Primary VID delete advertisement messages are exchanged among the switches and the switches reach a stage where the Primary VID table becomes empty. This is when the switches will actually switch to the Secondary Root.

Since there are twice as the number of switches in the previous topology, the average re-convergence time also increased from 0.017 seconds(Table 4.1) to 0.039 seconds in Table 4.3. The finding that stood out in this 10 Node topology experiment is the root election time for different tests in RSTP. The increased complexity added inconsistencies to the root election times for RSTP as highlighted in Table 4.3. The RSTP re-convergence time average also increased from 1.98 seconds to 3.039 seconds and it is attributed to the extra overhead of synchronizing the new root information to more number of switches in the 10 Node topology.

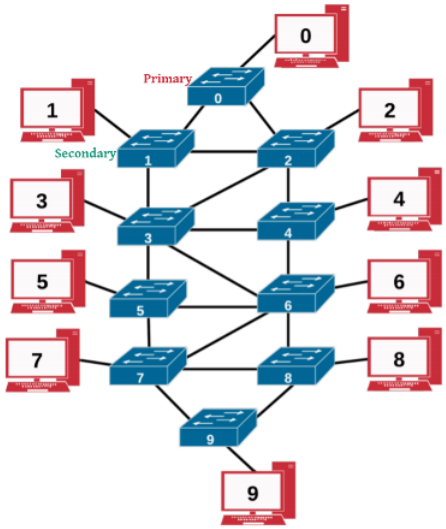

Figure 4.3: 10 Node Topology to illustrate frame loss

As shown in Figure 4.3, end node 2 is sending 1000 frames and when the root is down, the frame loss is calculated for all the other end-nodes. The slight increase in the convergence time resulted in a few extra frames loss in the 10 Node topology. The number of frames lost for each end node was dependent on the position in the

Primary Tree of the switch that it is connected to. Those end-nodes that did not have Node-0 in their tree path to the source(end-node-2) saw very minimal frame loss compared to the end-nodes that had Node-0 in their path. Looking at Figure 4.3, frames travelling from end-node 2 to end nodes connected to Nodes 4,6,8,9 are not majorly effected by the failure of Node-0 as they do not travel via Node-0. Even for the end-nodes that had frames lost, MMTP had an average of 151.3 frame loss where as RSTP frame loss was soaring high at around 698.3 frames.

4.3 17 Node Topology Results

The formation of Primary and Secondary VID tables is shown in the below Figure 4.4

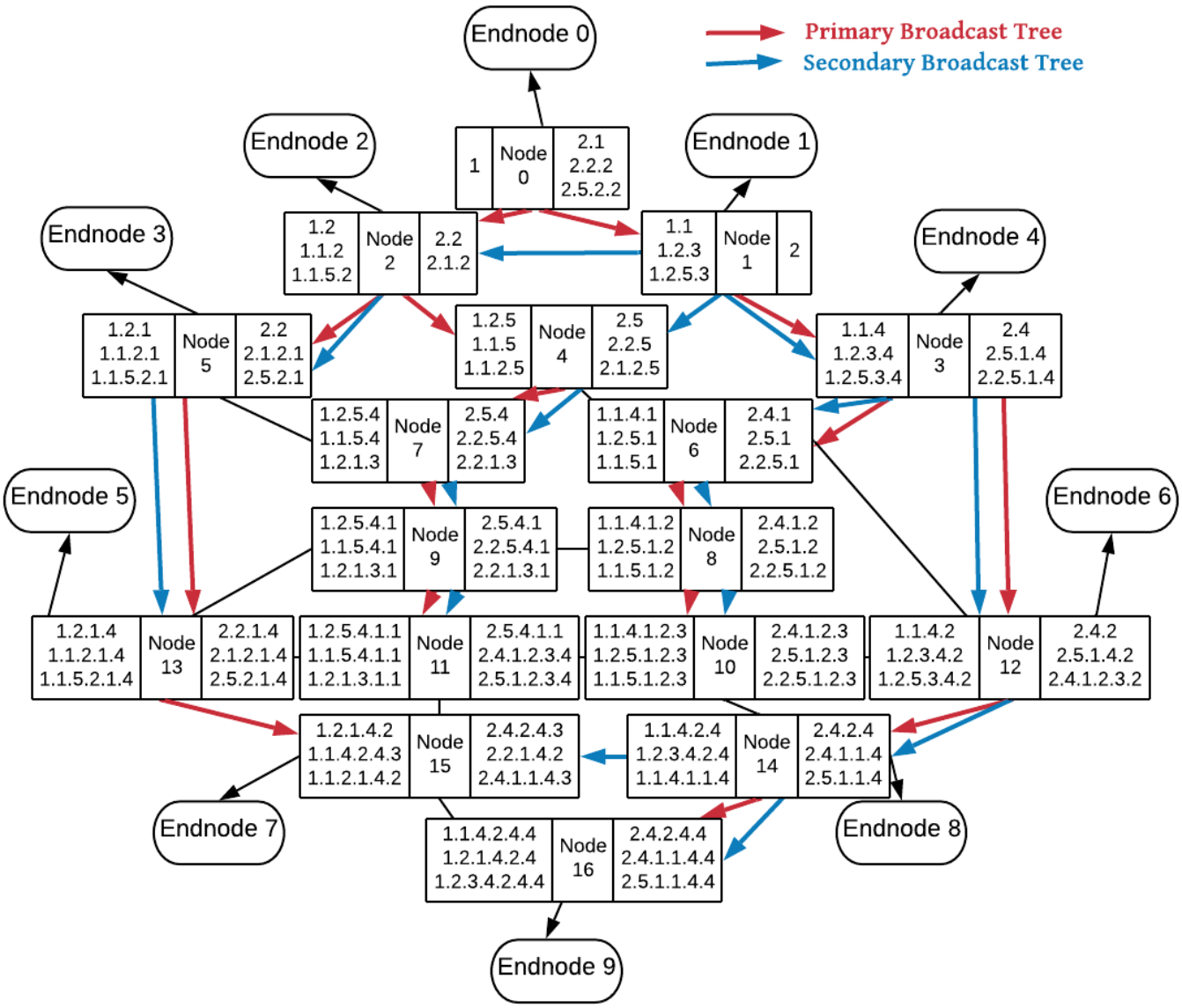

Figure 4.4: A Stable 17 Node Topology running the enhanced MMTP protocol

4.3.1 17-Node Convergence time comparison between RSTP vs MMTP

Table 4.5: 17-Node Convergence times(in seconds) comparison MMTP vs RSTP

Test#	Type of Delay	MMTP	RSTP
Test 1	Detection Time	0.91	5.380
	Root Election Time	0	20.252
	Re-convergence Time	0.18	4.023
	Total Convergence	1.09	29.655
Test 2	Detection Time	0.704	4.665
	Root Election Time	0	15.41
	Re-convergence Time	0.051	4.819
	Total Convergence	0.754	24.894
Test 3	Detection Time	0.938	4.624
	Root Election Time	0	24.607
	Re-convergence Time	0.055	6.040
	Total Convergence	0.994	35.271
Average	Detection Time	0.851	4.890
	Root Election Time	0	20.090
	Re-convergence Time	0.095	4.961
	Total Convergence	**0.946**	**29.940**

4.3.2 Loss of frames comparison between RSTP vs MMTP

Table 4.6: End Node 2 sending 1000 frames

Test#		End Node 2	End Node 1	End Node 3	End Node 4	End Node 5	End Node 6	End Node 7	End Node 8	End Node 9
Test 1	MMTP	121	3	121	3	121	3	121	0
	RSTP	478	0	478	0	478	0	478	0
Test 2	MMTP	199	2	2	2	199	0	199	199
	RSTP	626	0	626	0	626	0	626	0
Test 3	MMTP	197	5	197	5	197	5	197	0
	RSTP	591	0	591	0	591	0	591	0
Average	**MMTP**	**172.3**	**3.3**	**106.6**	**3.3**	**172.3**	**2.6**	**172.3**	**66.3**
	RSTP	**565**	**0**	**565**	**0**	**565**	**0**	**565**	**0**

4.3.3 17 Node Results Discussion

The largest and most meshed topology of the three, the 17 node topology results solidified the processes found in both protocols. The re-convergence times followed the same pattern as the 10 Node topology as the size of the topology increased, the re-convergence times also increased in both the topologies. For MMTP, it increased from an average of 0.039 to 0.095 seconds and for RSTP it increased from an average of 3.039 to 4.961. The results for the 17-Node topology exposed the complete impact of root failure on the RSTP protocol. As the topology grew and the complexity increased, the root election time went up to an average of 20.09 seconds. The reason for the long root election times might be due to the *count to infinity*

problem and the *race condition* problems in RSTP as discussed in section 2.3. As the topologies are getting bigger and complex, the variation in the overall convergence time for MMTP was very minimal varying from 0.748 seconds for a simple 5 Node topology to 0.946 for the complex 17 Node topology. Whereas for RSTP the problems increased as the topology got bigger and complex. The overall convergence for RSTP varied from 9.353 seconds for a simple 5 Node topology to 29.940 seconds for the complex 17 Node topology. This huge variation in the overall convergence for RSTP is attributed to the high root election times which varied from an average of 2.886 seconds for a simple 5 Node topology to 20.09 seconds for the complex 17 Node topology. This influencial delay is completely eliminated in MMTP and hence the results show a much superior performance of MMTP over RSTP.

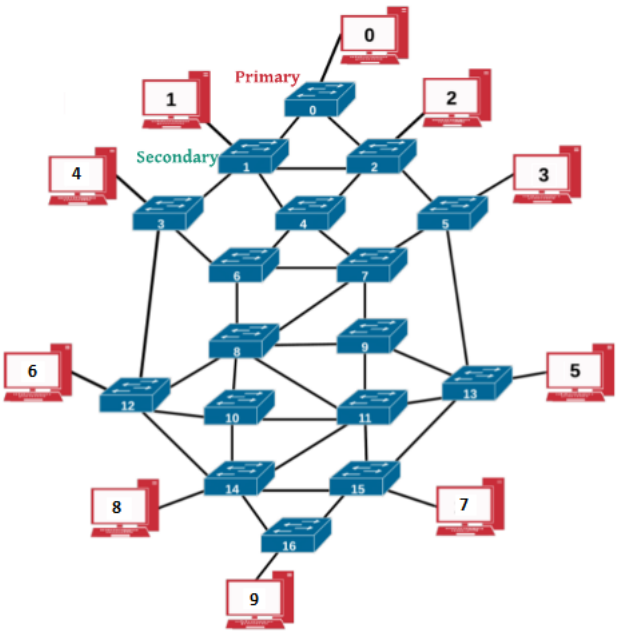

Figure 4.5: 17 Node Topology to illustrate frame loss

As shown in Figure 4.3, end-node 2 is sending 1000 frames and when the root is down, the frame loss is calculated for all the other end-nodes. The same number of end-nodes used in 10 Node topology are used in the 17 Node topology too. The number of frames lost for each end node was dependent on the position in the Primary Tree of the switch that it is connected to. Those end-nodes that did not have Node-0 in their tree path to the source(end-node-2) saw very minimal frame loss compared to the end-nodes that had Node-0 in their path. Looking at Figure 4.3, frames travelling from end-node 2 to end nodes connected to Nodes 3,5,7,9 are not effected by the failure of Node-0 as they do not travel via Node-0. Even for the end-nodes that had frames lost, MMTP had a maximum average of 172.3 frame loss where as RSTP frame loss was much high at around 565 frames.

Chapter 5

Conclusion

Mesh Tree Protocol is a great advancement in the field of study to avoid loops in meshed networks. Given the extensive research in the industry to find a replacement for the loop avoidance protocols in the market, Mesh Tree Protocol can be a potential replacement to the RSTP protocol that is widely used in the industry currently. This statement is further strengthened by the recent studies claiming the superior performance of MTP over RSTP. MTP's biggest advantage over the traditional widely used loop avoidance protocols is the use of logically built multiple trees ready to take over in cases of link failures. The next big advantage for Mesh Tree Protocol is the avoidance of the root election procedure and allowing the user to take the decision of choosing the root. This cuts down a considerable time of electing the root and also gives the advantage to choose the desired switch as root. This advantage comes with a huge dependency on the root and the threat for the root failure is prevalent. This thesis, introduces root redundancy into the Mesh Tree Protocol and allows the user to make his choice of primary root and secondary root. The threat of root failure no longer exists in the Mesh Tree Protocol as there is always a secondary tree existing in parallel and is always ready for fail-over. The root redundancy introduced in this thesis improves the reliability and availability of in networks running the Mesh Tree Protocol thereby strengthening the protocol's

overall robustness and its potential to compete against the current protocols that are widely used in the industry. After this thesis, MTP can be treated as a much robust protocol than it was before this thesis.

5.1 Future Work

This thesis is a proof of concept that the Mesh Tree Protocol is not vulnerable to root bridge failures and there can be multiple mesh trees originating from different backup roots. The user is given the choice of a secondary root and this study can be extended in the future to a third root or even more roots there by having more backup trees to fail-over. Also, this thesis did not explore the possibility of the primary root after the fail-over comes back and joins the network. Currently, when the primary root fails, the protocol switches to a secondary tree and continues to work based on the secondary tree and the secondary root. When the primary root comes back, it acts like a non-root to the secondary tree. Future studies can add a feature to roll back to the primary root when it comes back and is healthy. The added feature makes the protocol even more better aiding the user's original choice of picking the desired switch as primary root.

Bibliography

[1] IEEE Standard for Local Area Network MAC (Media Access Control) Bridges. *ANSI/IEEE Std 802.1D, 1998 Edition*, 1998. doi:10.1109/IEEESTD.1998.95619.

[2] IEEE Standard for Information Technology -Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Common Specifications - Part 3: Media Access Control (MAC) Bridges: Amendment 2 - Rapid Reconfiguration. *IEEE Std 802.1w-2001*, 2001. doi:10.1109/IEEESTD.2001.93287.

[3] IEEE Standards for Local and Metropolitan Area Networks - Amendment to 802.1Q Virtual Bridged Local Area Networks: Multiple Spanning Trees. *IEEE Std 802.1s-2002 (Amendment to IEEE Std 802.1Q, 1998 Edition)*, 2002. doi:10.1109/IEEESTD.2002.94223.

[4] IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, 2004. doi:10.1109/IEEESTD.2004.94569.

[5] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 20: Shortest Path Bridging. *IEEE Std 802.1aq-2012 (Amendment to IEEE Std*

802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, and IEEE Std 802.1Qbf-2011), 2012. doi:10.1109/IEEESTD.2012.6231597.

[6] MMTP implementation and testing Automation Scripts. URL: https://github.com/shashankrudroju/Thesis/tree/master/MTP_Automation.

[7] MMTP implementation code repository. URL: https://github.com/shashankrudroju/MTP-Root-Redundancy.

[8] Jeff Forcier. Paramiko, 2019. URL: http://www.paramiko.org/index.html.

[9] GNU Screen. URL: https://www.gnu.org/software/screen/.

[10] IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011) - IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks. URL: https://standards.ieee.org/findstds/standard/802.1Q-2014.html.

[11] K. Sharma, B. Hartpence, B. Stackpole, D. Johnson, and N. Shenoy. Meshed tree protocol for faster convergence in switched networks. In *2014 IARIA International Conference on Networking and Services (ICNS)*, pages 90–95, April 2014.

[12] Bill S. Daryl J. Kuhu S., Bruce H. and Nirmala S. Meshed Tree Protocol for Faster Convergence in Switched Networks. *IARIA and INRIA (Europe) sponsored Ninth International Conference on Networking and Services*, pages 20–24, 2014.

[13] Petr Lapukhov. Understanding STP and RSTP Convergence. *Internetwork Expert Inc.*, .

[14] Linux Foundation. Open vswitch, 2019. URL: `https://www.openvswitch.org/`.

[15] G. Prytz. Network recovery time measurements of rstp in an ethernet ring topology. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1247–1253, Sep. 2007. `doi:10.1109/EFTA.2007.4416924`.

[16] R. Pallos, J. Farkas, I. Moldovan, and C. Lukovszki. Performance of rapid spanning tree protocol in access and metro networks. In *2007 Second International Conference on Access Networks Workshops*, pages 1–15, Aug. 2007. `doi:10.1109/ACCESSNETS.2007.4447112`.

[17] Geni: Exploring networks of the future. URL: `https://www.geni.net/`.

[18] Routing bridges (rbridges): Base protocol specification july 2011. URL: `https://tools.ietf.org/html/rfc6325`.

[19] Understanding Rapid Spanning Tree Protocol (802.1w). URL: `http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/24062-146.html`.

[20] Kuhu Sharma. Implementation of the meshed tree algorithm on a switched network. Master's thesis, Rochester Institute of Technology, 2016.

[21] N. Shenoy and Y. Pan. Multi-meshed tree routing for internet manets. In *2005 2nd International Symposium on Wireless Communication Systems*, pages 140–149, Sep. 2005. doi:10.1109/ISWCS.2005.1547674.

[22] Understanding Spanning-Tree Protocol Topology Changes. URL: https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/12013-17.html.

[23] Standard Contributors. P1910.1 - standard for meshed tree bridging with loop free forwarding, 2019. URL: https://standards.ieee.org/project/1910_1.html.

[24] The Scapy Community. Scapy, 2019. URL: https://scapy.net/.

[25] The Wireshark team. Wireshark, 1998–. URL: https://www.wireshark.org/.

[26] Transparent Interconnection of Lots of Links (TRILL), May 2014. URL: https://tools.ietf.org/html/rfc7177.

[27] P. Willis and N. Shenoy. The Evaluation of a Meshed Tree Protocol on the GENI Testbed. *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–2, 2018.

[28] P. Willis and N. Shenoy. A meshed tree protocol for loop avoidance in switched networks. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 303–307, Feb 2019. doi:10.1109/ICCNC.2019.8685664.

[29] Peter Willis. A Performance Analysis of the Meshed Tree Protocol and the Rapid Spanning Tree Protocol. *Master of Science in Networking and Systems Administration from Rochester Institute of Technology*, 2019.

Vita

Shashank Rudroju was born in Hyderabad, India on June 19, 1991, the son of Ramesh Rudroju and Lalitha Rudroju. He received the Bachelor of Engineering degree in Electronics and Communication Engineering from Jawaharlal Nehru Technological Institute, Hyderabad, India in 2013. He is currently pursuing his Master of Science degree from Rochester Institute of Technology, United States of America. His research interest includes Networking, Data Analytics and Machine Learning. His current research includes root failure analysis in Mesh Tree Networks.

Permanent address: **Hyderabad, India**

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.